xAPP
XML Application Platform

Framework de desarrollo multiplataforma basado en la nube

Javier Berlana Hernández

Escuela Politécnica Superior

Universidad Carlos III de Madrid

Proyecto de Fin de Carrera

Ingeniería Informática

Octubre 2012

Título: xAPP: XML Application Platform.
Framework de desarrollo de aplicaciones móviles multiplataforma basado en la nube.

Autor: Javier Berlana Hernández
Director: Jorge Blasco Alís

El tribunal

1. Presidente: Agustín Orfila
adiaz@inf.uc3m.es

2. Vocal: Esther Palomar
epalomar@inf.uc3m.es

3. Secretario: Alejandro Calderón
acaldero@arcos.inf.uc3m.es

Realizado el acto de Defensa y Lectura del Proyecto Fin de Carrera el día 17 de octubre de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la calificación de

MATRÍCULA DE HONOR

Abstract

Actualmente existen diferentes frameworks de desarrollo multiplataforma para dispositivos móviles, que tratan de acelerar el proceso de creación de aplicaciones y facilitar la publicación simultánea en dispositivos con diferente sistema operativo.

Pero todos tienen el mimso inconveniente, y es que si se quieren obtener resultados semejantes a los que se esperan de una app nativa, o la curva de aprendizaje es alta, y por tanto difícil de encontrar personal cualificado para realizar el desarrollo, o los resultados no son buenos. Esto ocurre con los frameworks basados en HTML5, donde una persona familiarizada con el desarrollo web podría afrontar el proyecto, pero los resultados que se obtienen se acercan más a una aplicación web que a una app nativa para smartphones.

En este contexto nace la idea de crear xAPP, XML Application Platform, del inglés: Plataforma de Aplicaciones en XML. Un framework de desarrollo móvil multiplataforma que permite obtener aplicaciones nativas a partir de un modelo de aplicación sencillo definido en XML, en el que se puede desarrollar desde la nube, sin requerir entornos de desarrollo como Xcode o Eclipse, y que además permite desplegar en dispositivos de diferentes plataformas usando un sistema over-the-air (OTA).

Este Proyecto de Fin de Carrera detalla:

- La definición del lenguaje intermedio basado en un sistema de marcado XML y Javascript con el que se implementan las aplicaciones (en adelante xApps),
- La construcción de las librerías en Objective-C para la creación de aplicaciones para dispositivos con sistema operativo iOS, y
- La arquitectura cliente-servidor para la programación y distrubución de las xApps desde la nube.

Estoy seguro de que estos seis años han sido los más duros de mi vida. Los seis años en los que más cosas he aprendido. Los que me han convertido en la persona que soy hoy.

Quiero dar las gracias a mi familia, en especial a mis padres, a mi hermano Jorge y a mi abuelo José, quienes me han animado a perseverar para llegar hasta aquí. Sin su insistencia y empuje no hubiese terminado nunca este Proyecto.

Gracias a Puli, Chaves y Andrés, pues sin ellos los primeros años no hubiesen sido lo mismo. A Peter, mi fiel compañero de prácticas y mejor amigo, por esas tardes de hacking for fun and profit y por todos los proyectos que hemos empezado juntos. A Martín por los proyectos buenos y malos, por los que dan pasta, los proyectos con los que se aprende. A Diego, Charly, Juanto, Lalla, Judith, Pereira, Durán, Jesús, Jose, Virgi, Lillo, Chumo, Raúl, Xavi, Mario, Huete, Fombe, Chafi, Fer... seguro que me dejo a muchos, gracias por hacerme menos duros estos años y por todos los geniales momentos que hemos pasado dentro y fuera de la Universidad.

Gracias a toda la gente del Erasmus en Helsinki, por darme uno de los mejores años de mi vida.

También tengo que dar las gracias a todos los profesores que se esfuerzan por que nadie salga nunca de sus clases sin aprender algo nuevo, los que no se limitan a seguir el guion y recitar lo que dicen sus diapositivas. Me llevo muy buen recuerdo de Linares, Calderón, Calle, Olaya, Julio Cesar y alguno más.

Gracias a mis compañeros de Sweetbits, por contar conmigo para este proyecto.
Gracias a Isma, por su ayuda con la maquetación del portal web de xAPP.

Gracias a Jorge, mi tutor, un tío genial que creo ha sabido entender mejor que nadie lo difícil que ha sido para mi compaginar este Proyecto con el trabajo y el resto de proyectos en los que estoy involucrado.

Este es el final de una etapa que no voy a olvidar jamás.
Gracias a todos por compartirla conmigo.

The only way to do great work is to love what you do. If you haven't found it yet, keep looking. Don't settle. As with all matters of the heart, you'll know when you find it.

Steve Jobs (24 de febrero de 1955 - 5 de octubre de 2011)

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

En este capítulo se explicarán los motivos por los que he decidido llevar a cabo este Proyecto de Fin de Carrera, los objetivos del mismo y la estructura que se ha seguido al redactar esta memoria.

1.1 Motivación

En 2007, con la aparición del primer iPhone, se inició un cambio radical en el modo en el que las empresas y usuarios hacen uso del teléfono móvil; se pasó de llevar un aparato en el bolsillo con el que realizar llamadas, a tener un dispositivo con el que podemos hacer de todo; desde consultar el correo, reservar habitación en un hotel, ver qué restaurantes hay cerca y reservar mesa, o hacer una foto desde el propio terminal y compartirla con tus amigos a través de las redes sociales.

Muchas de estas posibilidades ya las ofrecían anteriormente las llamadas PDAs (Personal Digital Assistant), pero su uso estaba muy ligado al empresario y persona de negocios, no al usuario medio. Con la aparición de los smartphones el uso de las PDAs se fue reduciendo hasta desaparecer, al igual que la barrera que separa el tipo de dispositivo que usa una persona ligada al mundo de los negocios y un usuario de la calle.

Durante los últimos años el uso de smartphones se ha extendido exponencialmente, pasando de ser algo exclusivo y caro a estar masificado, especialmente en nuestro país. Tanto es así que en Europa España ostenta el primer puesto en cuanto a utilización de smartphones, con un 55,2%[1] del total de propietarios de móviles usando este tipo de dispositivos. Este creciemiento del mercado móvil sigue en alza, eso es algo que las grandes empresas saben y cada vez buscan tener mayor presencia en este tipo de dispositivos.

La aparición de nuevos dispositivos y la gran variedad de oferta ha ocasionado una gran fragmentación en el campo del desarrollo móvil. A día de hoy existen 4 Sistemas Operativos móviles diferentes: iOS de Apple, Android de Google, WindowsPhone de Microsoft y BlacBerry de RIM. Esto es algo genial para el usuario, ya que a mayor competencia mejores precios, mayor rivalidad y por tanto más rápido aparece tecnología nueva y puntera. Pero desde el punto de vista de las grandes empresas, el hecho de que existan 4 plataformas diferentes y tan diferenciadas provoca que los costes de desarrollo de aplicaciones sea muy caro, y más, si se quiere tener presencia en todas ellas.

Es en este punto donde cobra sentido la aparición de Frameworks de desarrollo que permitan crear aplicaciones móviles multiplataforma, sin la necesidad de disponer de diferentes equipos de desarrollo encargados de programar una misma aplicación para cada uno de los sistemas operativos móviles disponibles. Reduciendo de esta manera los tiempos y costes de desarrollo de una aplicación móvil.

[1] Según un estudio realizado por la compañía comScore sobre la situación de los principales mercados móviles europeos

1.2 Descripción general

xAPP, XML Application Platform, es una estructura de soporte definida, en la cual un proyecto de software para dispositivos móviles puede ser organizado y desarrollado directamente desde la nube.

xAPP pretende ser una herramienta para desarrollar aplicaciones para dispositivos móviles de una manera rápida y sencilla, de forma que un único desarrollo pueda ser ejecutado en múltiples plataformas. Además, pretende que este desarrollo no sea dependiente de la plataforma en la que se está desarrollando; es decir, que se pueda programar desde: Windows, Mac, Linux o incluso una tableta independientemente del sistema operativo.

El Proyecto está dividido en tres partes:

- El núcleo del Framework, que será lo que compartan las librerias de todas las plataformas; esto es: la grámatica del código XML con el que se desarrollan las aplicaciones y su parseo, los generadores de modelos de datos, vistas y el interprete de código javascript. Y además, las interfaces de comunicación entre las librerías cliente y el servidor. Todo esto, aunque portado al lenguaje de cada plataforma (Objective-C en iOS, Java en Android...) será común.

- Los elementos de las aplicaciones (cómo botones, vistas...) que servirán para representar en cada plataforma los elementos definidos de forma única en el XML. Aunque el resultado final, por ejemplo, que se vea un botón en el centro de la pantalla, será el mismo en todas sus plataformas la implementación será radicalemnte diferente para cada una de ellas.

- El portal web, desde dónde se crean y editan las aplicaciones (xApps) y se muestran los logs de estás.

A lo largo del documento se hará referencia a los dos primeros puntos cómo 'librería nativa', ya que es código que corre en el cliente, los smartphones, y es dependiente de cada plataforma. Por motivos de tiempo para este Proyecto de Fin de Carrera solo se ha desarrollado la librería nativa para la plataforma iOS.

1.2.1 Capacidades generales

- xAPP permite registrar usuarios desde un portal web (actualmente en http://jberlana.es/xapp).

- Desde este portal el usuario puede crear y editar sus aplicaciones móviles.

- Una aplicación de xAPP, en adelante xApp, tiene tres partes: vistas, modelo de datos y código, los tres pueden ser editados de la web.

- xAPP cuenta con la capacidad de notificar mediante PUSH a todos los dispositivos que tengan instalada una determinada aplicación cuando esta sea modificada.
- Desde el dispositivo es posible descargar todas las aplicaciones que han sido desarrolladas en xAPP.
- Una vez cargada una xApp en un dispositivo móvil esta puede enviar logs a través de internet para que el desarrollador tenga constancia del estado de la misma, pudiendo solucionar errores en caso de darse.
- xAPP permite manejar aplicaciones complejas con un modelo de datos complejo.
- xAPP tiene un interprete de código Javascript que permite a las xApps realizar operaciones complejas.
- La versión actual del framework permite crear aplicaciones completamente funcionales con muchas menos líneas de código de las que requeriría hacerlo de forma nativa.

1.2.2 Restricciones generales

- Actualmente solo existe librería nativa para iOS, por lo que el Framework no se puede utilizar en otras plataformas.
- La funcionalidad de las aplicaciones que se desarrollen con xAPP dependen del número y calidad de elementos de los que disponga el framework. En la versión actual son: botones, vistas, *labels*, campos de texto, tablas, mapas, imágenes y *WebViews*.
- Es necesario actualizar el Framework y añadir elementos nuevos cada vez que un sistema operativo móvil se actualiza para poder aprovechar nuevas funcionalidades.
- xAPP requiere de conexión a internet en el momento del login y para actualizar las aplicaciones instaladas.

1.3 Estructura de la memoria

Esta memoria ha sido dividida en 6 capítulos diferentes, los primeros enfocados a las fases de desarrollo de la plataforma y, los últimos, a plasmar las tareas de gestión realizadas para construir este proyecto y las conclusiones que se pueden extraer del mismo.

- Introducción: Capítulo que termina con esta sección y en el que se ha tratado de exponer por qué y para qué se ha creado xAPP.

- Análisis: En esta fase se analizarán los principales competidores; se expondrán las tecnologías utilizadas y el catalogo de requisitos del Framework.
- Diseño: Se mostrará la arquitectura y los componentes que forman el sistema.
- Implementación: Cómo y por qué se han desarrollado cada uno de los módulos del Framework junto a algunas aplicaciones de ejemplo desarrolladas con el mismo.
- Gestión: Planificaciones y costes.
- Conclusión y lineas futuras.
- Apéndices: Manuales de usuario y ejemplos.

Capítulo 2

Análisis

En este capítulo se realizará un estudio de los diferentes Frameworks de desarrollo multiplataforma que se encuentran al día de hoy en el mercado, comparando sus ventajas e inconvenientes y estudiando cómo xAPP puede mejorarlos.

Posteriormente se analizarán las herramientas, lenguajes, frameworks y librerias que se utilizarán en las etapas de desarrollo de xAPP.

2.1 Estado del arte

A día de hoy existen diferentes frameworks de desarrollo que permiten programar en un único lenguaje obteniendo apps que pueden ser ejecutadas en dispositivos con sistemas operativos diferentes. En todos los casos la solución pasa por realizar el desarrollo de estas aplicaciones utilizando o basandose en tecnologías web. A estas herramientas se les conoce cómo "write once, run everywhere", escribe una vez, ejecútalo en cualquier sitio.

El resultado obtenido al realizar el desarrollo de ésta manera, nunca será equiparable al desarrollo de una app nativa para cada una de las plataformas, ya que lo que esteremos haciendo es construir una web que haciendo uso de avanzados frameworks Javascript, HTML5 y hojas de estilo CSS3, de el aspecto dé aplicación nativa.

Haciendo uso de estos frameworks en un proyecto que requiera realizar una aplicación explotable desde múltiples plataformas móviles que en el caso más común serían: iPhone (ObjectiveC), Android (Java) y BlackBerry (Java), el desarrollo se limitará a realizar una app web adaptada a pantallas pequeñas, en lugar de mantener tres entornos de desarrollo diferentes para cada una de las plataformas, en caso contrario sería necesario el trabajo de varios equipos de desarrollo cada uno experto en una tecnología, dado que en iOS y Android el desarrollo de una aplicación es completamente diferente.

Las principales desventajas son:

- No hay acceso completo a todas las APIs nativas del móvil.
- Es lento: renderizar HTML e interpretar Javascript es sin duda más costoso que ejecutar una aplicación nativa, cada petición que hagamos en nuestra aplicación implicará una recarga de la página o un acceso, en mayor o menor medida, contra nuestro servidor. Aunque esto se podría suplir embebiendo los contenidos web en la propia aplicación en lugar de conectar con un servidor, de manera que al igual que una aplicación nativa, tenga todos los recursos y procesos guardados en local, y solo accediendo al servidor para obtener o enviar datos si es que los necesita.

Las dos plataformas más usadas actualmente a la hora de desarrollar una aplicación crossplatform son Appcelerator y PhoneGap, ambas basadas en el uso de tecnologías web.

Antes de describir cada una de estas haremos un pequeño repaso a los entornos de desarrollo de iOS y Android para porder entender mejor las diferencias entre el desarrollo nativo y el desarrollo *crossplatform*:

2.1.1 Entornos de desarrollo nativo

2.1.1.1 iOS

El lenguaje oficial para IOS es ObjectiveC. Hay distintas versiones de iOS pero todas ellas se programan usando ObjectiveC, y la misma herramienta, Xcode.

Xcode es el entorno de desarrollo oficial de Apple. Con él podemos crear aplicaciones de escritorio para Mac y para iOS. La mayoría de alternativas para crear aplicaciones en iPhone (Appcelerator, Phonegap, Corona) se apoyan siempre en esta herramienta para hacer el build final, aunque no todas (Flex no lo necesita).

El único "problema" que tiene es que solo existe para Mac, por lo que para crear aplicaciones iOS es necesario disponer de un ordenador de Apple.

Requisito importante: para distribuir aplicaciones en el App Store y para poder probar las aplicaciones desarrolladas en un dispositivo físico, es necesario adquirir una licencia de desarrollador que cuesta 80€al ano.

2.1.1.2 Android

En el caso de Android existen menos limitaciones, el lenguaje que se utiliza para desarrollar es Java junto a un SDK multiplataforma que funciona en Windows, Linux y Mac. Se puede dasarrollar con un IDE haciendo uso del plugin ADT para Eclipse, que incluye un simulador. Es multiplataforma, libre y gratuito.

Para publicar en el Market de Google debemos pagar una licencia de 5€pero, a diferencia de la licencia de Apple, ésta es una licencia de por vida.

2.1.2 Otros frameworks de desarrollo móvil

2.1.2.1 Appcelerator

Con Appcelerator[1] es posible crear aplicaciones para Android y iPhone usando exclusivamente Javascript (el soporte para Blackberry está en fase beta). Para programar proporciona Titanium Studio, un IDE basado en Eclipse con el que crear y manejar los proyectos. Experimentalmente permite usar Php, Ruby y Python, pero siempre transformando posteriormente el código a Javascript con los frameworks Phpjs, Skulpt o Ruby.js.

Las aplicaciones se programan integramente con Javascript, creando y colocando "a mano" todos los controles. Para ello se hace uso de una librería interna de Titanium que hace de envoltorio entre nuestra aplicación Javascript y los controles del sistema.

[1] Appcelerator: http://www.appcelerator.com/

Esto significa que las ventanas y demás controles visuales (botones, listas, menus, etc) son nativos; cuando se añade un botón esta librería crea un botón del sistema y lo añade a la vista, por lo que renderiza más rápido y la respuesta del usuario es también mejor que la de una webApp. A diferencia de PhoneGap, en Appcelerator no hay DOM, por lo que no se pueden usar librerías como jQuery ya que el contexto de ejecución es Javascript puro, no dentro de un documento HTML.

Una de las características más interesantes de Appcelerator (y que más confusión produce, debido a que se vende como un generador de aplicaciones nativas), es que al empaquetar la aplicación, el código Javascript es transformado y compilado, pero no a Objective-C si no a un lengual intermedio que después, cuando se arranca la aplicación en el móvil, será ejecutado dentro de un engine Javascript. JavaScriptCore en iOS (el intérprete de Webkit, el motor de Safari y Chrome) y Mozilla Rhino en Android/Blackberry. Si fuera compilado y ejecutado exclusivamente en Objective-C, como mucha gente piensa erróneamente, no sería posible utilizar las capacidades dinámicas que tiene el lenguaje Javascript en tiempo de ejecución.

El hecho de que el código Javascript esté compilado y que los controles creados sean nativos, le hace tener el mejor rendimiento posible (teniendo en cuenta que es Javascript, claro) en comparación con PhoneGap o Adobe Air.

Con Appcelerator es complicado maquetar, pues no existe un HTML inicial donde añadir los controles, sino que hay que crear las ventanas y componentes con Javascript utilizando el API de Titanium. También es posible crear un único componente Webkit a pantalla completa, y trabajar de manera parecida a como lo hace PhoneGap, explicado en la seccion 2.1.2.4. Por otro lado, puede que algo que se desarrolle para iPhone no funcione para Android, y viceversa. Los desarrollos de las librerías Javascript evolucionan por separado y hay que mirar bien qué se puede hacer y cómo. A diferencia de PhoneGap, que solo tiene una librería Javascript para acceder a las características especiales del sistema, Appcelerator necesita además librerías para manejar los controles nativos y su disposición en la pantalla, por lo que el desarrollo en general es más costoso.

Sobre el soporte para iPhone/iPad, es necesario tener Mac con Xcode 4 instalado. Lo que hace Titanium Studio es generar un IPA utilizando comandos de Xcode para compilar desde terminal, este IPA contiene todo el Javascript transformado junto con las librerías necesarias. Después es posible lanzar el simulador con la aplicación sin salir de Titanium Studio.

Sobre el soporte Android, tanto para probar en el simulador como para empaquetar la aplicación, solo hay que tener el SDK de Android instalado.

Ventajas:

- Multiplataforma móvil y también de escritorio.

- Aspecto y controles nativos. El mejor rendimiento.
- Gratis, soporte de pago. Licencia Apache.

Desventajas:

- Requiere Mac y Xcode para empaquetar aplicaciones iOS.
- Definición de componentes visuales y controles "a mano" (PhoneGap es HTML y Flex es MXML)
- Mucha documentación pero poco actualizada y descolocada, tutoriales desfasados, poco profesional.

2.1.2.2 Frameworks HTML 5

A la hora de desarrollar en HTML 5 tenemos varios frameworks que facilitan y aceleran el desarrollo, además aplican un estilo a la página que le da el aspecto de app nativa. Este documento se centrará en jQuery Mobile y Sencha Touch.

Tanto JQuery Mobile[1] cómo Sencha[2] son frameworks Javascript que tratan de mover al ecosistema móvil la labor que desempeñan sus "hermanos mayores", jQuery y ExtJs, en el entorno de las aplicaciones web.

A la vista de los ejemplos de cada una de las webs (http://jquerymobile.com/demos/1.0/ frente a http://www.sencha.com/products/touch/demos/) y la toma de contacto que se ha realizado con cada una de las plataformas, se podría decir que haciendo uso de Sencha Touch se obtienen mejores resultados, aplicaciones que dan un aspecto más similar al que podría tener una aplicación desarrollada de manera nativa, cosa que no ofrece una app desarrollada haciendo con jQuery Mobile. Por contrapartida programar usando Sencha requiere ser experto en el Framework ExtJS, framework que se caracteriza por tener una gran robustez y estabilidad.

Los puntos más destacables de cada uno de éstos Frameworks son los siguientes:

Sencha Touch

- Su forma de uso es muy similar a ExtJS. En caso de conocer y haber trabajado con ExtJS será sencillo hacerlo con Sencha Touch. En caso contrario no es un sistema fácil.
- Contiene temas para el diseño de la interfaz muy vistosos y de simple configuración.

[1] JQuery Mobile: http://jquerymobile.com/
[2] Sencha Touch: http://www.sencha.com/products/touch/

Figura 2.1: **Sencha Touch** -

- Tiene un API muy completo así como buena documentación.
- Manejo y uso de Ajax directamente controlado por el framework.
- Basado en HTML 5

jQuery Mobile

- No requiere escribir mucha cantidad de código JavaScript para lograr aplicaciones móviles. Al escribirse menos se logran resultados realmente rápidos.
- En lugar de orientarse a la programación JavaScript, jQuery mobile se centra en usar etiquetas HTML con atributos definidos por el framework y al momento de mostrar la página en el cliente obtiene estas informaciones y crea lo necesario para mostrar los componentes.
- Al igual que Sencha Touch, contiene temas para el diseño de la interfaz muy vistosos y de simple configuración.
- Manejo y uso de Ajax directamente controlado por el framework.
- Basado en HTML 5

2.1.2.3 Flex, Adobe Air mobile

Adobe Air mobile funciona con Flex 4 y soporta las plataformas iOS, Android y BlackBerry Tablet.

Flex 4 utiliza el lenguaje de programación ActionScript, que es (por hacer una comparación) un Javascript compilado, de fuerte tipado y orientado a objetos con el que

Figura 2.2: **jQuery Mobile** -

poder hacer complejos desarrollos. El IDE oficial, Flash Builder 4.5, es un IDE muy potente, pero de pago. Es posible compilar y empaquetar las aplicaciones con Flex SDK, opensource y gratuito (basado en Java), aunque con Flash Builder es mucho más fácil e inmediato, ya que proporciona una gran cantidad de wizards y editores.

Los controles visuales usados durante el desarrollo y ejecución no son los originales de cada plataforma, sino que son específicos de Flex 4. Esto tiene su lado bueno y su lado malo: en su ejecución los elementos no van tan "suaves" como si fueran nativos, pero nos garantiza que todas las aplicaciones tendrán exactamente el mismo aspecto y comportamiento.

Una de las peculiaridades es que es la única herramienta que no requiere ni Android SDK ni Xcode para Mac para ejecutar y crear las aplicaciones.

Ventajas:

- Multiplataforma móvil y también de escritorio.
- ActionScript es un lenguaje muy potente que permite el uso de patrones y estructuras complejas en los desarrollos.
- Desarrollo y definición de las vistas "a golpe de ratón" con el editor visual de MXML con Flash Builder. En general, el IDE y Flex 4 están muy avanzados y son muy potentes.
- La documentación es bastante completa y fácil de comprender.
- Flash Builder 4.5 no requiere el uso de Xcode ni Mac.

Desventajas:

- El precio de Flash Builder 4.5 ($699). Aunque hay otras herramientas y se puede usar el SDK gratuito.
- No funciona en todos los Android, solo en los de gama alta que tengan arquitectura Arm7.
- Rendimiento regular, renderización de componentes no muy suave en iOS.
- Aspecto no nativo (aunque homogéneo entre todas las plataformas).

2.1.2.4 Phonegap

PhoneGap es un sistema para crear aplicaciones usando exclusivamente HTML5, CSS3 y Javascript, que son ejecutadas dentro en un componente WebKit. Provee una serie de librerías Javascript desarrolladas en el lenguaje específico de cada plataforma (Objetctive-C para iOS, Java para Android, etc) que nos permiten acceder a las características del móvil como GPS, acelerómetro, cámara, contactos, base de datos, filesystem, etc. Al ser una página web, tenemos acceso al DOM y podemos usar frameworks web como jQuery o cualquier otro.

Requiere diseñar tu aplicación web con los componentes visuales típicos de HTML o usar un framework web mobile como jQuery Mobile o Sencha Touch para darle una apariencia nativa. Tiene la ventaja de que puedes definir la navegación inicial de la aplicación usando Chrome o Firefox de tu ordenador, sin tener que ejecutarla en el simulador.

En resumen, podemos ver una aplicación PhoneGap como una serie de páginas web que están almacenadas y empaquetadas dentro de una aplicación móvil por medio de un navegador web, pero que nos permite acceder a la mayoría de Apis del móvil, lo cual lo convierte en una alternativa muy sencilla para crear aplicaciones con gran funcionalidad.

Para trabajar con cada plataforma hay que usar un sistema distinto: para iPhone/iPad es necesario usar Xcode y una plantilla de proyecto que proporciona PhoneGap. Para Android se debe usar Eclipse con su respectiva plantilla de proyecto. Y para Blackberry no hay entorno: solo Java SDK, BlackBerry SDK y Apache Ant.

En la figura 2.3 se puede ver cuales son las APIs soportadas por PhoneGap en cada tipo de dispositivo:

Ventajas:

- Es la solución que más plataformas móviles soporta, pues corre dentro de un navegador web. Además de iPhone/iPad y Android, funciona también en Palm,

	iPhone / iPhone 3G	iPhone 3GS and newer	Android	OS 4.6-4.7	OS 5.x	OS 6.0+	WebOS	WP7	Symbian	Bada
ACCELEROMETER	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
CAMERA	✓	✓	✓	✗	✓	✓	✓	✓	✓	✓
COMPASS	✗	✓	✓	✗	✗	✗	✗	✓	✗	✓
CONTACTS	✓	✓	✓	✗	✓	✓	✗	✓	✓	✓
FILE	✓	✓	✓	✗	✓	✓	✗	✓	✗	✗
GEOLOCATION	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
MEDIA	✓	✓	✓	✗	✗	✗	✗	✓	✗	✗
NETWORK	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (ALERT)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (SOUND)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
NOTIFICATION (VIBRATION)	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
STORAGE	✓	✓	✓	✗	✓	✓	✓	✓	✓	✗

Figura 2.3: **Tabla de funcionalidad** - La figura muestra las APIs disponibles con Phonegap dependiendo de la plataforma. Esta tabla ha sido extraida de la documentación oficial de Phonegap en http://phonegap.com/

Symbian, WebOS, W7 y BlackBerry,

- Es muy fácil de desarrollar y proporciona una gran libertad a los que tienen conocimientos de HTML y Javascript.
- Hay buena documentación y bastantes ejemplos.
- Es gratis, soporte de pago. Licencia BSD.

Inconvenientes:

- La aplicación no es más que una página web, por lo que el aspecto dependerá del framework web utilizado. Necesitaremos el uso de frameworks HTML móviles como Sencha Touch, jQuery mobile, Jo, Sproutcore, XUI o jQTouch entre otros si queremos que parezca una aplicación nativa.
- No llega al rendimiento de una aplicación nativa, pues el HTML, CSS y Javascript debe ser leído e interpretado por el engine del navegador cada vez que arranca.

2.2 Análisis de las tecnologías utilizadas

En esta sección se describen las tecnologías utilizadas para llevar a cabo cada uno de los módulos de los que se compone el Proyecto.

2.2.1 XML

XML, siglas en inglés de eXtensible Markup Language ('lenguaje de marcas extensible'). Ha sido el lenguaje utilizado para la descripción tanto de las vistas, cómo del modelo de datos de las xApps.

El motivo del uso de esta tecnología es que, a diferencia de otros lenguajes, XML da soporte a bases de datos, siendo útil cuando varias aplicaciónes se deben comunicar entre sí o integrar información; además, se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable ajustándose perfectamente a los requisitos de xAPP y solucionando uno de los primeros retos: La representación de la estructura gráfica de una xApp, de manera que distintas plataformas la puedan interpretar.

2.2.2 JSON

JSON es la tecnología elegida en la capa de comunicación de las xApps con el servidor. El motivo de elegir JSON, en lugar de XML, para este cometido es que XML es complejo y está bastante sobrecargado, lo que hace que a la hora de transmitir mensajes estos sean más pesados, ocupando mayor ancho de banda y provocando transmisiones más lentas.

JSON, al tratarse de un lenguaje más simple, dinámico, compacto y permisivo que XML lo hace mejor candidato para la transmisión de mensajes; además, al contrario que XML los mensajes no necesitan ser validados, lo que le da puntos frente a XML a la hora de transmitir datos, pero se los quita en el caso de la estructuración de información y es por eso por lo que no ha sido elegido para la descripción de vistas y modelo de datos de las xApps.

Además, existen numerosas herramientas para parsear ficheros JSON muy potentes y eficientes; en el caso de iOS, desde iOS 5, el SDK cuenta con un parser de JSON nativo, para versiones anteriores se usan librerías cómo JSONKIT [1]

2.2.3 SQLite y MySQL

Estos han sido los sistemas gestores de bases de datos utilizados, SQLite cómo sistema de persistencia de datos en la librería cliente y MySQL en el servidor.

[1] Web de JSONKit: https://github.com/johnezang/JSONKit

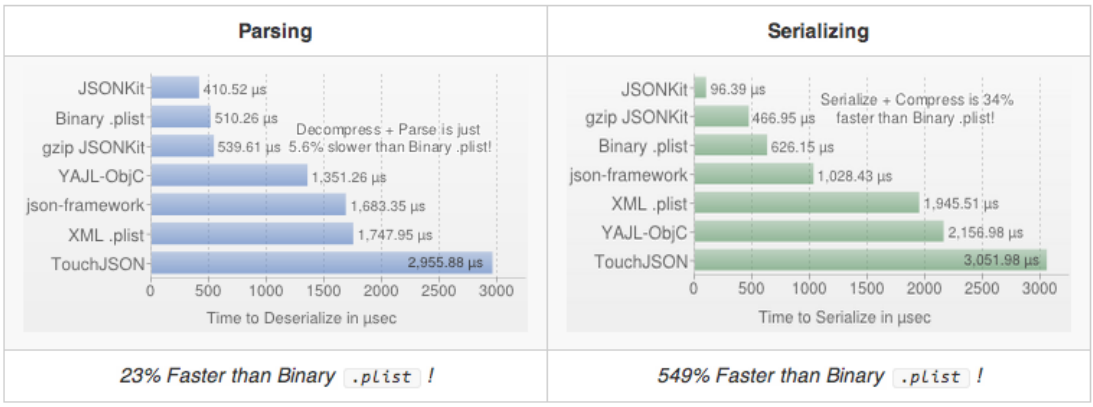

Figura 2.4: **Rendimiento de JSONKIT** - La figura muestra el rendimiento de JSONKit frente a otras libreías para procesar JSON en Objective-C. Datos extraidos de http://www.bonto.ch/blog/2011/12/08/json-libraries-for-ios-comparison-updated/

A la hora de elegir una tecnología de persistencia para el desarrollo de la librería nativa para iOS podemos hacerlo entre dos opciónes, usar CoreData o SQLite, CoreData ofrece multitud de ventajas frente a SQLite, en lo que a rendimiento se refiere; además desde la aparición de iCloud con iOS 5 tenemos la posibilidad de mantener todos nuestro esquema de persistencia en la nube.

Pero CoreData se encuentra únicamente disponible en iOS, y xAPP pretende ser un Framework de desarrollo multiplataforma. Por lo que para poder compartir la mayor parte del código posible en el desarrollo de librerías nativas he optado por elegir SQLite, que se encuentra disponible en Android y Windows Phone.

2.2.4 PHP

PHP ha sido el lenguaje de programación utilizado para la creación del sitio web del framework, se trata de un lenguaje de programación interpretado diseñado para la creación de páginas web dinámicas.

Se usa tanto para la interpretación del lado del servidor (server-side scripting) a la hora de generar las páginas dinámicas que se le muestran al usuario, cómo en el API, para recibir las llamadas que llegan desde los dispositivos móviles y proporcionar respuestas en JSON.

2.2.5 AFNetworking

AFNetworking[1] ha sido la librería usada para la comunicación entre la librería iOS y el servidor. Se trata de una librería de red para iOS y Mac OS X construida sobre NSURLConnection, NSOperation y otras tecnologías Fundation.

Tiene una arquitectura modular con buen diseño, lo que lo hace especialmente potente y fácil de integrar con aplicaciónes iOS. La alternativa a usar AFNetworking era ASIHTTPRequest[2], una librería similar pero que recientemente ha sido descontinuada por el autor.

2.2.6 Codemirror

CodeMirror[3] es un componente JavaScript utilizado en la web de xAPP para poder proporcionar un editor de código en el navegador.

Es compatible con un gran número de lenguajes, en el caso de xAPP basta con XML y Javascript, pero incluye muchos más. Permite el resaltado de código y funciones de autoindentado lo que ayudará a los usuarios a la hora de crear sus aplicaciones.

Dispone de una API amplia y bien documentado lo que ha ayudado a adaptarlo para que cumpla con los requisitos de usuario.

[1]Web de AFNetworking: https://github.com/AFNetworking/AFNetworking
[2]Web de ASIHTTPRequest: http://allseeing-i.com/ASIHTTPRequest
[3]Web de Codemirror: http://codemirror.net/

2.3 Catálogo de requisitos

En este apartado se detallará la especificación de los requisitos software, tanto funcionales como no funcionales, que debería cumplir el sistema a desarrollar.

Por cada uno de los requisitos se especifican los siguientes campos:

- Identificador: El identificador determina de forma unívoca cada uno de los requisitos.
- Descripción: Breve descripción de la funcionalidad o restricción que dicho requisito engloba.
- Necesidad: Indica si el requisito tiene que ser implementado de forma obligatoria u opcional. Los valores que puede tomar este campo son:
  - Esencial: El requisito debe ser implementado.
  - Deseable: Es importante implementar el requisito pero no obligatorio.
  - Opcional: Es irrelevante la implementación del requisito. Se podrá implementar pero no es obligatorio.
- Prioridad: Indica la importancia del requisito en el proceso de diseñoo e implementación. Los valores que puede tomar este campo son:
  - Alta: El requisito debe ser añadido al sistema en primer lugar.
  - Media: El requisito debe ser añadido al sistema tras haber acabado con los requisitos de prioridad alta.
  - Baja: El requisito debe ser añadido al sistema tras haber acabado con los requisitos de prioridad media.

2.3.1 Características de los usuarios

En xAPP podemos encontrar tres tipos de usuarios.

- **Colaboradores.** El personal que altruistamente decida colaborar en el desarrollo y evolución de xAPP, en caso de que se decida abrir el proyecto y venderlo como open source colaborativo. Estos usuarios crearán nuevos módulos y portarán módulos existentes para que las aplicaciones se puedan ejecutar en otras plataformas.
- **Desarrolladores.** Son usuarios técnicos que desarrollarán aplicaciones móviles usando xAPP; éstos serán los usuarios directos de xAPP.
- **Usuarios finales.** Los usuarios que utilizarán las aplicaciones desarrolladas haciendo uso de xAPP.

2.3.2 Restricciones

Las resticciones de xAPP vendrán dadas por las limitaciones de cada plataforma y las librerías nativas desarrolladas para estas.

Las restricciones vendrán dadas por los lenguajes de programación de cada una de las plataformas para las que se programe la libreria nativa que permitan utilizar xAPP. En este sentido, Objective-C el lenguaje requerido en iOS, Java en Android, C# en Windows Phone.

No existen restricciones de Hardware.

2.3.3 Requisitos de Software

General de xAPP			
Identificador	RSW-1	Prioridad	Baja
		Necesidad	Esencial
Descripción			
xAPP, a través de su librería nativa, debe ser capaz de interpretar el código escrito en XML para mostrar la aplicación independientemente de la plataforma móvil que se esté utilizando.			

Tabla 2.1: Requisito de software 1.

General de xAPP			
Identificador	RSW-2	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework a desarrollar debe ser capaz de generar aplicaciones ejecutables en iOS			

Tabla 2.2: Requisito de software 2.

General de xAPP			
Identificador	RSW-3	Prioridad	Alta
		Necesidad	Esencial
Descripción			
xAPP debe permitir incluir código javascript en las aplicaciones, de manera que se puedan realizar tareas complejas.			

Tabla 2.3: Requisito de software 3.

General de xAPP			
Identificador	RSW-4	Prioridad	Media
		Necesidad	Esencial
Descripción			
El framework debe permitir a un desarrollador distribuir sus aplicaciones en multilples positivos, incluso de diferentes plataformas. Esta distribución podrá hacerse over-the-air, sin necesidad de requerir contacto directo con el usuario final.			

Tabla 2.4: Requisito de software 4.

General de xAPP			
Identificador	RSW-5	Prioridad	Media
		Necesidad	Opcional
Descripción			
Las aplicaciones construidas con el framework deben mostrar un aspecto similar o idéntico al de otras aplicaciones nativas de la plataforma.			

Tabla 2.5: Requisito de software 5.

General de xAPP			
Identificador	RSW-6	Prioridad	Baja
		Necesidad	Esencial
Descripción			
Las aplicaciones creadas con el framework deben poder explotar las carácterísticas del dispositivo móvil donde corran.			

Tabla 2.6: Requisito de software 6.

General de xAPP			
Identificador	RSW-7	Prioridad	Alta
		Necesidad	Opcional
Descripción			
El framework debe ser modulable permitiendo a otros desarrolladores añadir nuevos elementos a xAPP.			

Tabla 2.7: Requisito de software 7.

General de xAPP			
Identificador	RSW-8	Prioridad	Alta
		Necesidad	Esencial
Descripción			
Añadir elementos en el framework no debe afectar a aplicaciones ya desarrolladas.			

Tabla 2.8: Requisito de software 8.

General de xAPP			
Identificador	RSW-9	Prioridad	Media
		Necesidad	Esencial
Descripción			
Los elementos desarrollados deben compartir funcionalidad y aspecto entre plataformas.			

Tabla 2.9: Requisito de software 9.

General de xAPP			
Identificador	RSW-10	Prioridad	Media
		Necesidad	Esencial
Descripción			
Las aplicaciones instaladas deben poder ser eliminadas.			

Tabla 2.10: Requisito de software 10.

General de xAPP			
Identificador	RSW-11	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir que las aplicaciones tengan distintas ventanas, y controlar la navegación entre ellas.			

Tabla 2.11: Requisito de software 11.

General de xAPP			
Identificador	RSW-12	Prioridad	Media
		Necesidad	Opcional
Descripción			
El framework debe permitir personalizar el color de fondo de las páginas.			

Tabla 2.12: Requisito de software 12.

General de xAPP			
Identificador	RSW-13	Prioridad	Baja
		Necesidad	Opcional
Descripción			
El framework debe permitir utilizar degradados como fondo para las páginas.			

Tabla 2.13: Requisito de software 13.

General de xAPP			
Identificador	RSW-14	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir añadir un título a las páginas.			

Tabla 2.14: Requisito de software 14.

General de xAPP			
Identificador	RSW-15	Prioridad	Media
		Necesidad	Esencial
Descripción			
El framework debe permitir distribuir libremente los elementos en pantalla.			

Tabla 2.15: Requisito de software 15.

General de xAPP			
Identificador	RSW-16	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir la definición de toda la estructura de vistas en XML.			

Tabla 2.16: Requisito de software 16.

General de xAPP			
Identificador	RSW-17	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir la definición del modelo de datos de las aplicaciones.			

Tabla 2.17: Requisito de software 17.

General de xAPP			
Identificador	RSW-18	Prioridad	Media
		Necesidad	Opciona
Descripción			
El framework debe permitir definir objetos por página asociados a tablas del modelo de datos, editables por otros elementos o controladores.			

Tabla 2.18: Requisito de software 18.

General de xAPP			
Identificador	RSW-19	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir el desarrollo de las aplicaciones a través de un portal web.			

Tabla 2.19: Requisito de software 19.

Del portal web de xAPP			
Identificador	RSW-20	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El portal web debe permitir registrar nuevos usuarios.			

Tabla 2.20: Requisito de software 20.

Del portal web de xAPP			
Identificador	RSW-21	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El portal web debe permitir *logarse* a los usuarios.			

Tabla 2.21: Requisito de software 21.

Del portal web de xAPP			
Identificador	RSW-22	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El portal web debe controlar que un usuario no pueda acceder a las aplicaciones desarrolladas por otro.			

Tabla 2.22: Requisito de software 22.

Del portal web de xAPP			
Identificador	RSW-23	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El portal web debe permitir la creación de aplicaciones.			

Tabla 2.23: Requisito de software 23.

Del portal web de xAPP			
Identificador	RSW-24	Prioridad	Media
		Necesidad	Opciona
Descripción			
El portal web debe permitir la visualización de los logs generados por las aplicaciones.			

Tabla 2.24: Requisito de software 24.

Del portal web de xAPP			
Identificador	RSW-25	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El portal web debe permitir editar la arquitectura de vistas de las aplicaciones.			

Tabla 2.25: Requisito de software 25.

Del portal web de xAPP			
Identificador	RSW-26	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El portal web debe permitir editar el modelo de datos de las aplicaciones.			

Tabla 2.26: Requisito de software 26.

Del portal web de xAPP			
Identificador	RSW-27	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El portal web debe permitir editar el código de las aplicaciones.			

Tabla 2.27: Requisito de software 27.

Del portal web de xAPP			
Identificador	RSW-28	Prioridad	Baja
		Necesidad	Opcional
Descripción			
El editor de código del portal web deber permitir el resaltado y coloreado de código.			

Tabla 2.28: Requisito de software 28.

Del portal web de xAPP			
Identificador	RSW-29	Prioridad	Media
		Necesidad	Opcional
Descripción			
El editor de código del portal web debe permitir deshacer los cambios realizados, hasta un máximo de 40.			

Tabla 2.29: Requisito de software 29.

Del portal web de xAPP			
Identificador	RSW-30	Prioridad	Media
		Necesidad	Opcionl
Descripción			
El editor de código del portal web deber permitir rehacer los cambios deshechos.			

Tabla 2.30: Requisito de software 30.

Del portal web de xAPP			
Identificador	RSW-31	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El editor de código del portal web permitir guardar los cambios realizados.			

Tabla 2.31: Requisito de software 31.

Del portal web de xAPP			
Identificador	RSW-32	Prioridad	Media
		Necesidad	Esencial
Descripción			
El framework debe notificar a los dispositivos cuando se han realizado cambios en alguna de las aplicaciones instaladas.			

Tabla 2.32: Requisito de software 32.

Del portal web de xAPP			
Identificador	RSW-33	Prioridad	Baja
		Necesidad	Esencial
Descripción			
El framework debe permitir a un usuario eliminar sus aplicaciones.			

Tabla 2.33: Requisito de software 33.

De la librería especifica de cada plataforma			
Identificador	RSW-34	Prioridad	Alta
		Necesidad	Esencial
Descripción			
La librería nativa debe permitir hacer login a los usuarios registrados en la web.			

Tabla 2.34: Requisito de software 34.

De la librería especifica de cada plataforma			
Identificador	RSW-35	Prioridad	c
		Necesidad	d
Descripción			
La librería nativa debe mantener las credenciales de un usuario una vez a hecho login, de manera que no tenga que volver a logarse.			

Tabla 2.35: Requisito de software 35.

De la librería especifica de cada plataforma			
Identificador	RSW-36	Prioridad	Alta
		Necesidad	Esencial
Descripción			
La librería nativa debe permitir a un usuario descargar todas sus aplicaciones creadas en la web una vez a hecho login.			

Tabla 2.36: Requisito de software 36.

De la librería especifica de cada plataforma			
Identificador	RSW-37	Prioridad	Alta
		Necesidad	Esencial
Descripción			
La librería nativa debe permitir ejecutar las aplicaciones creadas por el usuario.			

Tabla 2.37: Requisito de software 37.

De la librería especifica de cada plataforma			
Identificador	RSW-38	Prioridad	Media
		Necesidad	Opciona
Descripción			
La librería nativa debe poder enviar al servidor los logs que produzcan las aplicaciones, si el dispositivo se encuentra conectado a internet.			

Tabla 2.38: Requisito de software 38.

De la librería especifica de cada plataforma			
Identificador	RSW-39	Prioridad	Media
		Necesidad	Opcional
Descripción			
La librería nativa debe ofrecer una capa de geolocalización a las aplicaciones desarrolladas.			

Tabla 2.39: Requisito de software 39.

De la librería especifica de cada plataforma			
Identificador	RSW-40	Prioridad	Media
		Necesidad	Opcional
Descripción			
La librería nativa debe mostrar avisos si se recibe una notificación de actualización.			

Tabla 2.40: Requisito de software 40.

De la librería especifica de cada plataforma			
Identificador	RSW-41	Prioridad	Media
		Necesidad	Esencial
Descripción			
La librería nativa debe poder actualizar las aplicaciones instaladas sin necesidad de reiniciar.			

Tabla 2.41: Requisito de software 41.

De la librería especifica de cada plataforma			
Identificador	RSW-42	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir almacenar datos en los dispositivos de manera que éstos datos sean persistentes entre usos de la aplicación.			

Tabla 2.42: Requisito de software 42.

De la librería especifica de cada plataforma			
Identificador	RSW-43	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir al desarrollador añadir botones a la aplicación.			

Tabla 2.43: Requisito de software 43.

De la librería especifica de cada plataforma			
Identificador	RSW-44	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir al desarrollador cambiar el títlo de los botones.			

Tabla 2.44: Requisito de software 44.

De la librería especifica de cada plataforma			
Identificador	RSW-45	Prioridad	Media
		Necesidad	Opcional
Descripción			
El framework debe permitir al desarrollador editar el color de los botones.			

Tabla 2.45: Requisito de software 45.

De la librería especifica de cada plataforma			
Identificador	RSW-46	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir al desarrollador añadir acciones a los botones.			

Tabla 2.46: Requisito de software 46.

De la librería especifica de cada plataforma			
Identificador	RSW-47	Prioridad	Media
		Necesidad	Esencial
Descripción			
El framework debe permitir al desarrollador modificar el tamaño de los botones.			

Tabla 2.47: Requisito de software 47.

De la librería especifica de cada plataforma			
Identificador	RSW-48	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir añadir cadenas de texto(Labels) a las vistas.			

Tabla 2.48: Requisito de software 48.

De la librería especifica de cada plataforma			
Identificador	RSW-49	Prioridad	Baja
		Necesidad	Esencial
Descripción			
El framework debe permitir al desarrollador modificar la fuente de los labels.			

Tabla 2.49: Requisito de software 49.

De la librería especifica de cada plataforma			
Identificador	RSW-50	Prioridad	Baja
		Necesidad	Opcional
Descripción			
El framework debe permitir al desarrollador modificar el color de los labels.			

Tabla 2.50: Requisito de software 50.

De la librería especifica de cada plataforma			
Identificador	RSW-51	Prioridad	Media
		Necesidad	Opcional
Descripción			
El framework debe permitir al desarrollador modificar la alineación de los labels.			

Tabla 2.51: Requisito de software 51.

De la librería especifica de cada plataforma			
Identificador	RSW-52	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir que otros elementos puedan editar las propiedades de un label.			

Tabla 2.52: Requisito de software 52.

De la librería especifica de cada plataforma			
Identificador	RSW-53	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir añadir campos de texto a las aplicaciones.			

Tabla 2.53: Requisito de software 53.

De la librería especifica de cada plataforma			
Identificador	RSW-54	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir que el texto de estos campos sea editable por el usuario final.			

Tabla 2.54: Requisito de software 54.

De la librería especifica de cada plataforma			
Identificador	RSW-55	Prioridad	Media
		Necesidad	Esencial
Descripción			
El framework debe permitir al programador editar el texto por defecto de éstos campos.			

Tabla 2.55: Requisito de software 55.

De la librería especifica de cada plataforma			
Identificador	RSW-56	Prioridad	Media
		Necesidad	Opcional
Descripción			
El framework debe permitir al programador editar la alineación del texto dentro de los textfields.			

Tabla 2.56: Requisito de software 56.

De la librería especifica de cada plataforma			
Identificador	RSW-57	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe poder mapear el contenido de un campo de texto con con un objeto de la página y con una tabla del modelo de datos.			

Tabla 2.57: Requisito de software 57.

De la librería especifica de cada plataforma			
Identificador	RSW-58	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir al desarrollador añadir imágenes a sus aplicaciones.			

Tabla 2.58: Requisito de software 58.

De la librería especifica de cada plataforma			
Identificador	RSW-59	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir cargar imagenes desde una URL de intenet.			

Tabla 2.59: Requisito de software 59.

De la librería especifica de cada plataforma			
Identificador	RSW-60	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir al desarrollador editar el tamaño de las imágenes.			

Tabla 2.60: Requisito de software 60.

De la librería especifica de cada plataforma			
Identificador	RSW-61	Prioridad	Media
		Necesidad	Opcional
Descripción			
El framework debe permitir al desarrollador editar la posición de las imagenes en pantalla.			

Tabla 2.61: Requisito de software 61.

De la librería especifica de cada plataforma			
Identificador	RSW-62	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir al desarrollador colocar mapas en las aplicaciones.			

Tabla 2.62: Requisito de software 62.

De la librería especifica de cada plataforma			
Identificador	RSW-63	Prioridad	Alta
		Necesidad	Esencial
Descripción			
Los mapas deben poder geoposicionar al usuario de la aplicación.			

Tabla 2.63: Requisito de software 63.

De la librería especifica de cada plataforma			
Identificador	RSW-64	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir convertir una dirección en coordenadas (georeferenciar).			

Tabla 2.64: Requisito de software 64.

De la librería especifica de cada plataforma			
Identificador	RSW-65	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir calcular la ruta entre dos puntos.			

Tabla 2.65: Requisito de software 65.

De la librería especifica de cada plataforma			
Identificador	RSW-66	Prioridad	Alta
		Necesidad	Esencial
Descripción			
El framework debe permitir al desarrollador añadir vistas web a sus aplicaciones, para cargar directamente una página web.			

Tabla 2.66: Requisito de software 66.

De la librería especifica de cada plataforma			
Identificador	RSW-67	Prioridad	Alta
		Necesidad	Esencial
Descripción			
Las vistas webs deben permitir navegar hacia atrás.			

Tabla 2.67: Requisito de software 67.

De la librería especifica de cada plataforma			
Identificador	RSW-68	Prioridad	Media
		Necesidad	Opcional
Descripción			
Las vistas webs deben permitir navegar hacia adelante cuando se haya ido hacia atrás.			

Tabla 2.68: Requisito de software 68.

De la librería especifica de cada plataforma			
Identificador	RSW-69	Prioridad	Alta
		Necesidad	Esencial
Descripción			
Las vistas webs deben permitir al desarrollador cargar una URL por defecto.			

Tabla 2.69: Requisito de software 69.

De la librería especifica de cada plataforma			
Identificador	RSW-70	Prioridad	Alta
		Necesidad	Esencial
Descripción			
Las vistas web deben ofrecer un API al desarrollador para que otros elementos de la aplicación puedan editar la URL cargada, o disparar eventos, como ir a la página anterior.			

Tabla 2.70: Requisito de software 70.

Capítulo 3

Diseño

Este capitulo trata de explicar cómo se ha diseñado xAPP justificando cada una de las decisiones tomadas en ésta etapa de desarrollo de la plataforma.

Se comenzará detallando los patrones utilizados en el diseño de la arquitectura; posteriormente se pasará al diseño de la plataforma en la nube y se terminará enumerando los componentes con los que cuenta xAPP en la versión actual.

3.1 Arquitectura del sistema

3.1.0.1 Modelo Vista Controlador (MVC)

Modelo Vista Controlador es un patrón o modelo de abstracción de desarrollo de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos.

- Modelo: Esta es la representación específica de la información con la cual el sistema opera. En resumen, el modelo se limita a lo relativo de la vista y su controlador facilitando las presentaciones visuales complejas. El sistema también puede operar con más datos no relativos a la presentación, haciendo uso integrado de otras lógicas de negocio y de datos afines con el sistema modelado.
- Vista: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.
- Controlador: Este responde a eventos, usualmente acciones del usuario, e invoca peticiones al modelo y, probablemente, a la vista.

Es importante prestar especial atención a este patron de disño ya que en el se fundamenta todo el diseño del Framework.

- MVC entre los dispositivos y el servidor, el controlador la librería nativa instalada en el dispositivo que transforma el XML en la aplicación resultante, que será la vista, y el modelo sería el código XML generado en el servidor.
- MVC en la propia definición de las xApps en XML dónde nos encontramos claramente con el código de las vistas, el modelo de datos que define objetos y tipos y el controlador en forma de código javascript.

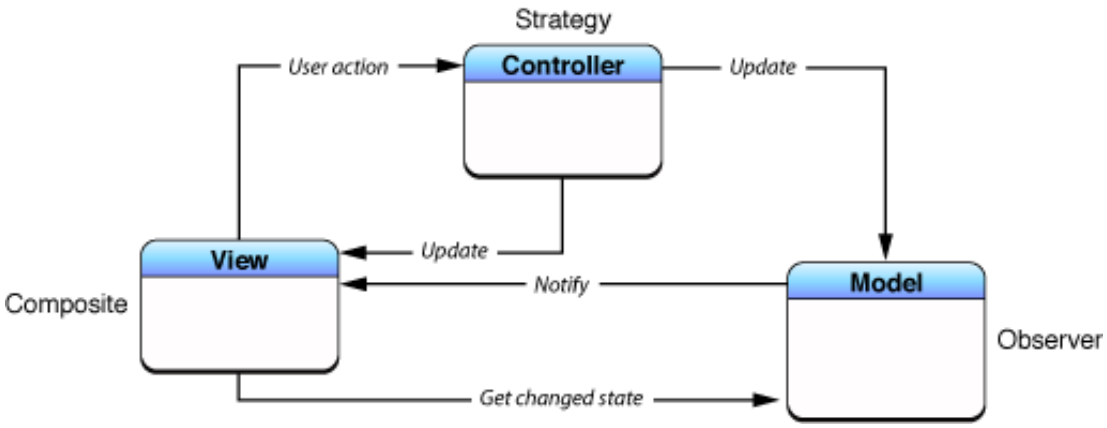

Figura 3.1: **Modelo Vista Controlador, imagen de la documentación oficial de Apple.** -

3.1.1 Los componentes del sistema

En esta sección se describirán los componentes que componen cada una de los elementos del Framework; éstos componentes se han dividido en tres partes;

- Cómo se ha diseñado la librería nativa para iOS y cada uno de los módulos que la componen.
- El IDE de desarrollo en la nube, desde dónde se programan las aplicaciones.
- Y por último. cómo está diseñada una xApp y las secciones de las que se compone.

Cada plataforma móvil necesitará implementar una versión de esta librería que inicialmente he desarrollado para iOS; su cometido es leer el código XML que define las aplicaciones y transformarlo en elementos nativos de la plataforma en tiempo de ejecución.

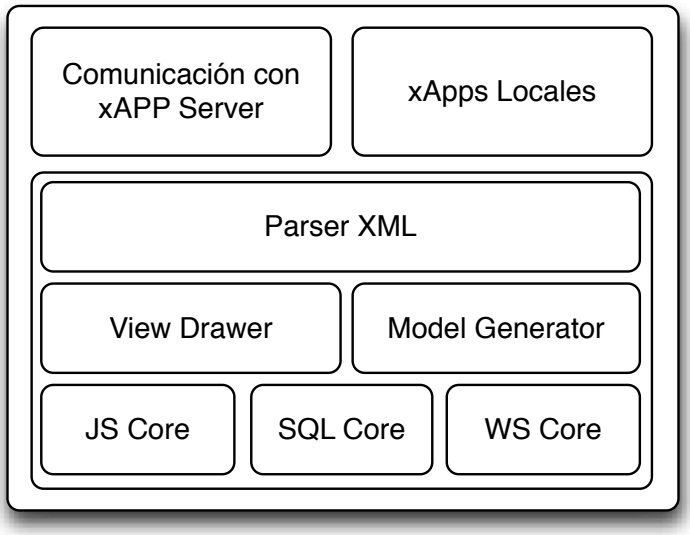

Figura 3.2: **Arquitectura de la librería nativa** -

Cómo se puede observar en la figura 3.2 el parser es uno de los elementos principales de la arquitectura, se encargará de leer el XML que define la aplicación y de generar dos estructuras de datos independientes: una con el árbol de vistas subvistas y elementos que compondrán cada una de las páginas de la aplicación y otra con el modelo de datos que posteriormente será transformado por el motor SQL en una base de datos SQLite.

Este servicio básico de lectura y interpretación de xApps es además apoyado por el módulo encargado de manejar la comunicación con el servidor, a través de las APIs expuestas, y el motor Javascript encargado de ejecutar el código de cada una de las xApps.

3.1.1.1 Parser XML

El parser XML es un parser de tipo SAX, las siglas de "Simple API for XML", originalmente, una API únicamente para el lenguaje de programación Java, que después se convirtió en la API estándar de facto para usar XML en JAVA.

Para entender cómo funciona el parser, y posteriormente como está estructurado el código de una xApp, conviene repasar cómo está estructurado un documento XML. Por definición, un documento XML es una cadena de caracteres. Casi todos los caracteres Unicode legal puede aparecer en un documento XML.

Los caracteres que componen un documento XML se dividen en marcas y contenido, los cuales se pueden distinguir por simples reglas sintácticas. En general, las cadenas que constituyen marcas o bien empezar por el carácter `<` y terminan con a `>`, o que comienzan con el carácter `&` y terminan con un`;`. El resto de cadenas de caracteres que no son marcas se denomina contenido.

En el grupo de las marcas nos encontramos con las etiquetas, una construcción de marcado que comienza con `<` y termina con `>`. Y pueden ser de tres tipos diferentes:

- Etiquetas iniciales, por ejemplo: `<content>`
- Etiquetas de fin, por ejemplo: `</ content>`
- Etiquetas vacías o de elemento, por ejemplo: `<line-break />`

3.1.1.2 SQL Core y Generador de modelos de datos

Este módulo se encarga, una vez parseado el XML de la aplicación, de crear un script SQL de creación para generar la base de datos de la aplicación. La base de datos es creada únicamente si no éxiste, pero si se detectan cambios en el modelo de datos de la xApp el generador de modelos es capaz de alterar el esquema definido en la creación de la base de datos.

La base de datos recibirá el nombre de la xApp, por tanto, y dado que el nombre de una xApp es identificador único, podremos tener tantas bases de datos cómo xApps haya instaladas en el dispositivo.

Una vez la base de datos se crea, se rellena con los datos definidos en el modelo, para ello se generan scripts de creación en tiempo de ejecución haciendo totalmente dinámico el acceso a la base de datos.

3.1.1.3 Generador de vistas

La Generador de Vistas es el elemento más importante de xAPP, ya que es el encargado de transformar la estructura XML que define la aplicación en una arquitectura de vistas

válida en la plataforma que ejecuta la xApp.

Este módulo, junto a los controladores y elementos, es el que hace de xAPP un Framework multiplataforma ya que modificando la forma en la que se generan todas las vistas y se añaden a éstas los diferentes elementos y controladores de manera aplicada a cada una de las plataformas bastaría para tener al app corriendo en diferentes dispositivos con un único código base.

El Generador de vistas funciona de manera recursiva para poder leer y generar las vistas de asociadas a un árbol cómo el que se muestra en la figura 3.10

El Generador de vistas, no sólo se encarga de pintar las vistas de la aplicación, si no que también responderá a los eventos producidos por cada uno de controladores y elementos que se encuentren en su página, haciendo de *manager* entre ellos y proporcionándoles un contexto para que puedan interactuar con el resto de elementos de la aplicación y con el modelo de datos.

En el diagrama de flujo de la figura 3.3 podemos observar cada una de las clases que entran en juego a la hora de cargar una xApp. En las siguientes secciones entraremos en detalle a definir qué se espera de cada una de ellas y cuál es su labor.

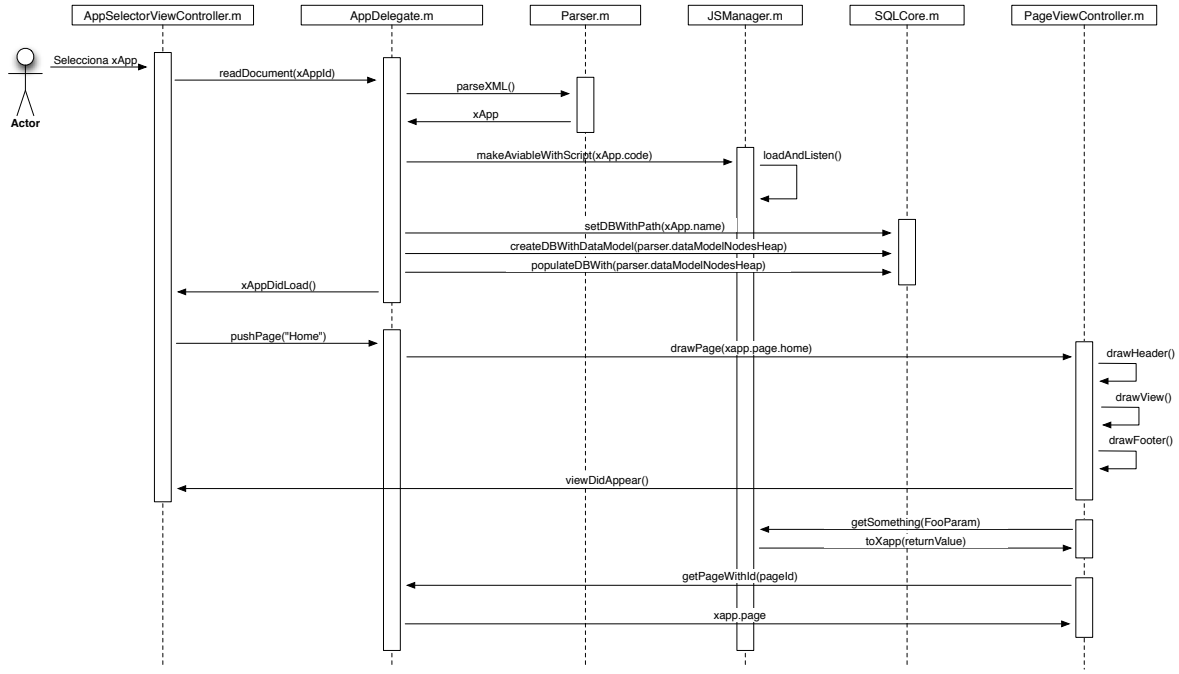

Figura 3.3: **Diagrama de flujo de carga de una xApp -**

Con el fin de poder añadir nuevos elementos y controladores al Framework sin que

sea necesario tocar las clases del núcleo del mismo, se hadiseñado un sistema deonde todas los nuevos elementos deberán extender de una clase base e implementar los métodos de su interfaz; haciendo esto podremos añadir elementos a nuestras xApps indicando en el tipo de los elementos incluidos en el XML el nombre que le hemos dado a las clases que estamos creando. Por ejemplo, si hemos creado `MyNewController.m` crearemos un elemento de este tipo con
`<element id=10 type=MyNewController>`

La figura 3.4 muestra como todos los componentes incluidos en el sistema extienden de `ElementView` e implementan el protocolo `ElementProtocol`.

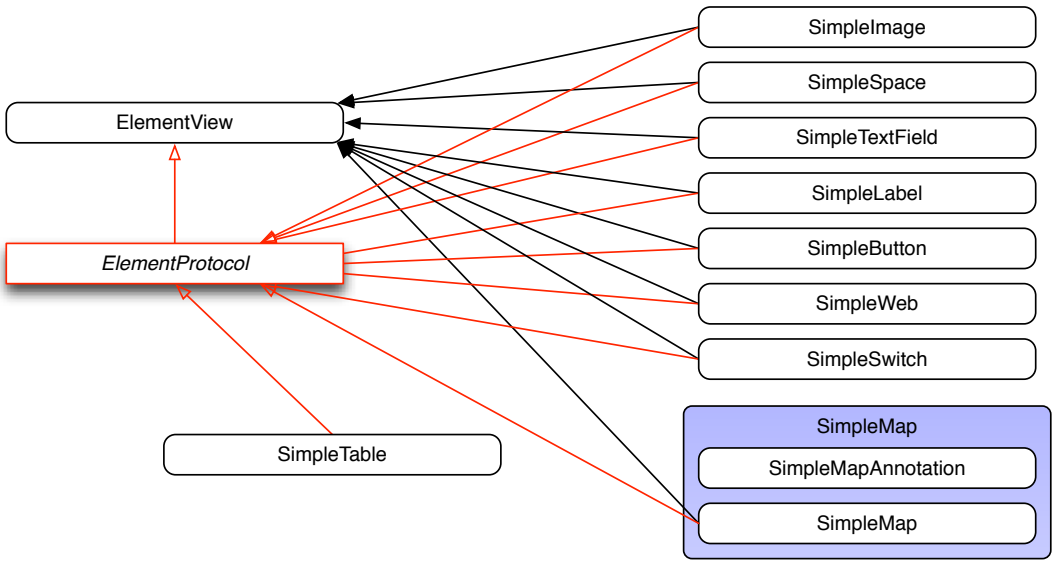

Figura 3.4: **Diagrama de clases** - Módulo de componentes

3.1.1.4 Núcleo Javascript

Este modulo se encargará de ejecutar el javascript de la aplicación y devolver la respuesta, se comunicará directamente con el delegado de la aplicación que quien se encargará de sincronizar las llamadas entre componentes.

En la figura 3.5 se observa un diagrama de clases con el diseño de los componentes. Por motivos de brevedad no se incluyen todas las clases ni las relaciones entre estas.

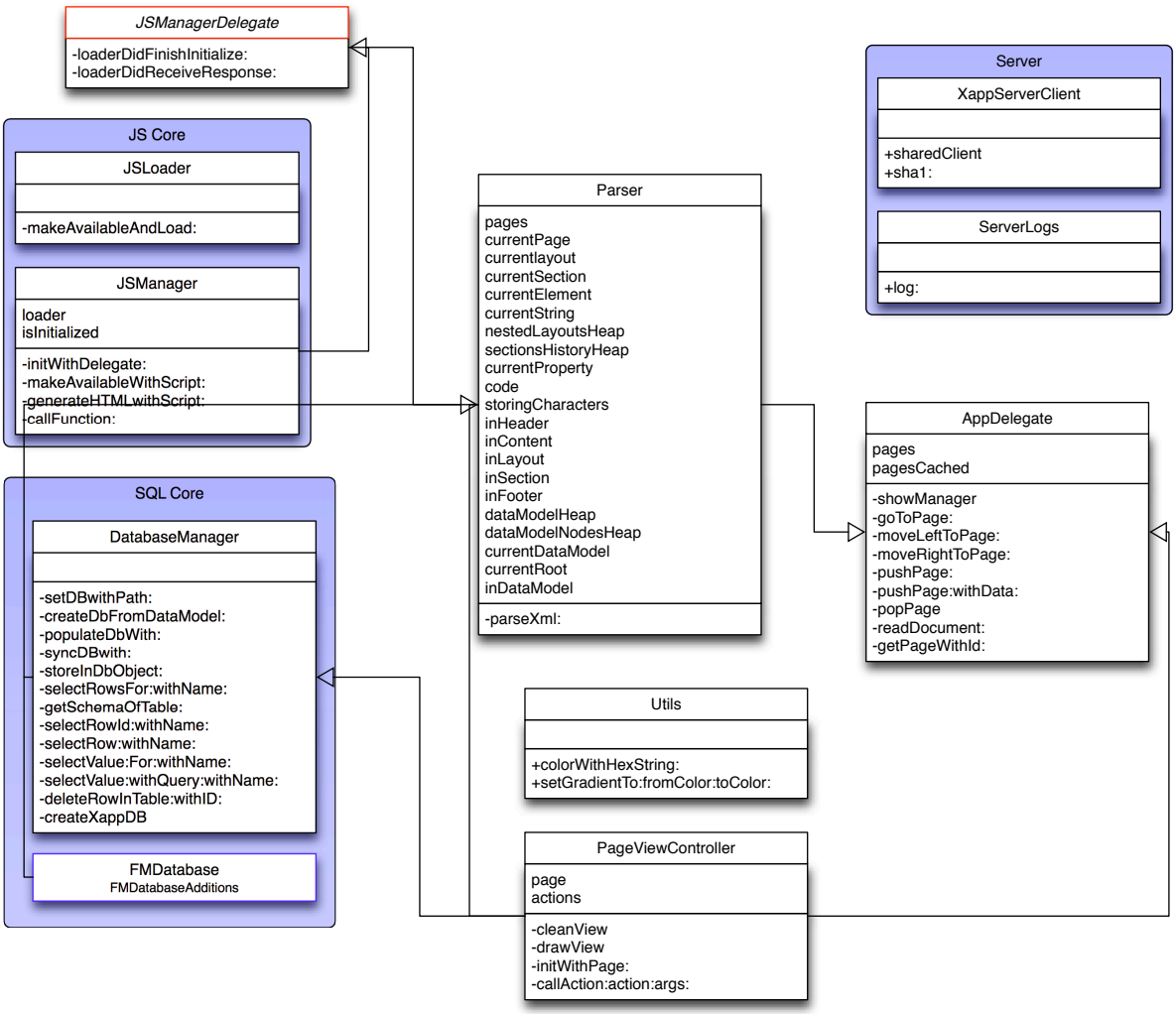

Figura 3.5: **Diagrama de clases del núcleo de xAPP -**

3.1.2 En la nube

Uno de los componentes más importantes e innovadores que incluye xAPP, es la posibilidad de desarrollar desde la nube. Para ello se ha desarrollado este elemento SaaS, software as a service que se estructura en tres módulos, el IDE web desde el que desarrollar, crear y editar las aplicaciones; los webservices expuestos que permiten la comunicación desde los módulos en la nube y los dispositivos móviles y el sistema de notificaciones Push. Todo esto esta soportado por un gestor de bases de datos MySql. En la figura 3.6 se puede observar cómo están estructurados estos módulos.

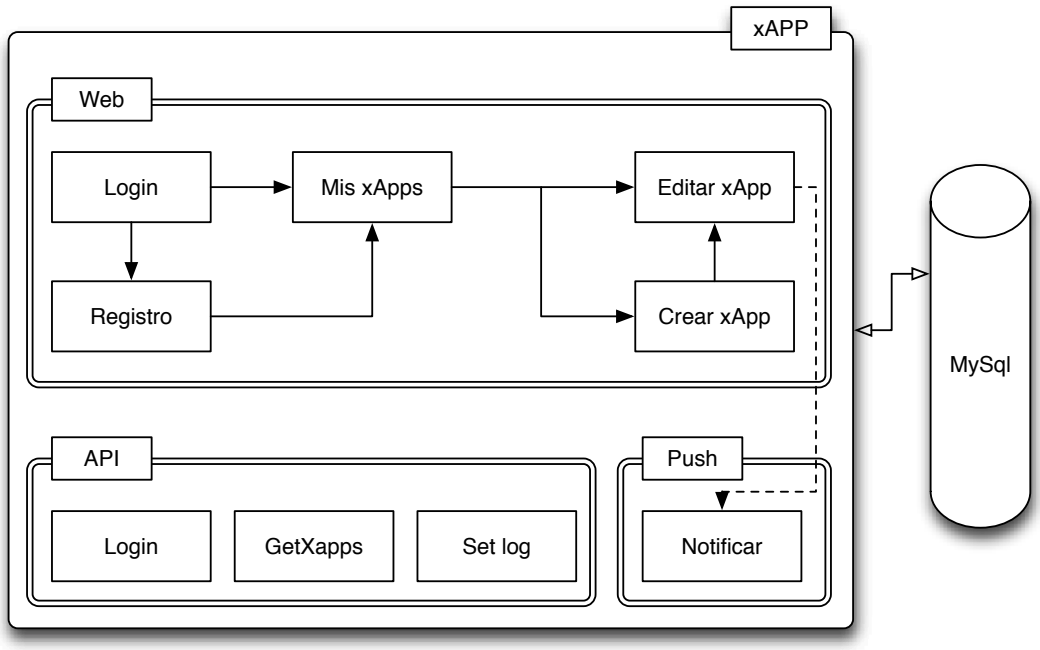

Figura 3.6: **Arquitectura en la nube** -

Por motivos económicos, y dado que la xAPP no se encuentra en producción, el servidor es un servidor no seguro, pero por motivos de seguridad es necesario disponer de un servidor https con certificados SSL para mantener un canal de comunicación seguro entre cliente y servidor.

3.1.2.1 Editor de código

xAPP permite crear y editar las xApps directamente desde la interfaz web, para ello utiliza una de las librerias Javascript más utilizadas en internet para realizar esta tarea,

Codemirror.

La interfaz web nos permite crear aplicaciones (xApps) dándoles un nombre y descripción, y posteriormente editarlas, modificando desde las vistas al modelo de datos pasando por el código Javascript que necesitasen para funcionar.

Alguna de las características que tiene el editor es la posibilidad de utilizar una pila para poder hacer y deshacer durante la edición, además cuenta con resaltado de sintaxis para xml y javascript.

Como veremos posteriormente xAPP cuenta con un sistema para notificar e instalar las aplicaciones directamente en los dispositivos móviles sin la necesidad de requerir cables o cuentas de desarrollador.

3.1.2.2 Control de errores

Dado que en xAPP se desarrolla desde la nube, no se cuentan con los mecanismos clásicos de debug con los que puede contar un IDE de desarrollo desde escritorio. Es por esto necesario aportar los mecanismos adecuado para mostrar al usuario por que la aplicación se ha cerrado inesperadamente o no arranca.

Para ello la librería nativa cuenta con un sistema de control de excepciones que envía trazas al servidor cuando se produce un error, bien al realizar el parseo del código XML de la xApp, o posteriormente durante su ejecución.

El servidor guarda la información básica de la excepción: traza, hora, xApp en la que se ha producido... Pero además guardará información estadística cómo: usuario que ha lanzado la excepción, plataforma desde la que se ha producido, versión del SO móvil...

3.1.2.3 Servicio de notificaciones

xAPP implementa un servicio de notificaciones PUSH para notificar a un dispositivo móvil cuándo se han producido cambios en alguna de las aplicaciones que tiene instaladas.

El servicio de notificaciones que implementa xAPP está basado en la tecnología Push, o servidor push, la cual describe un estilo de comunicaciones sobre Internet donde la petición de una transacción se origina en el servidor, al contrario de la tecnología Pull, donde la petición es originada en el cliente. De esta manera se consigue mandar el aviso incluso en casos donde el cliente esté cerrado cómo se puede observar en la figura A.3.

Más adelante, en el capítulo Implementación, se mostrará como se ha desarrollado el servicio de mensajes Push para el caso de la librería nativa para iOS.

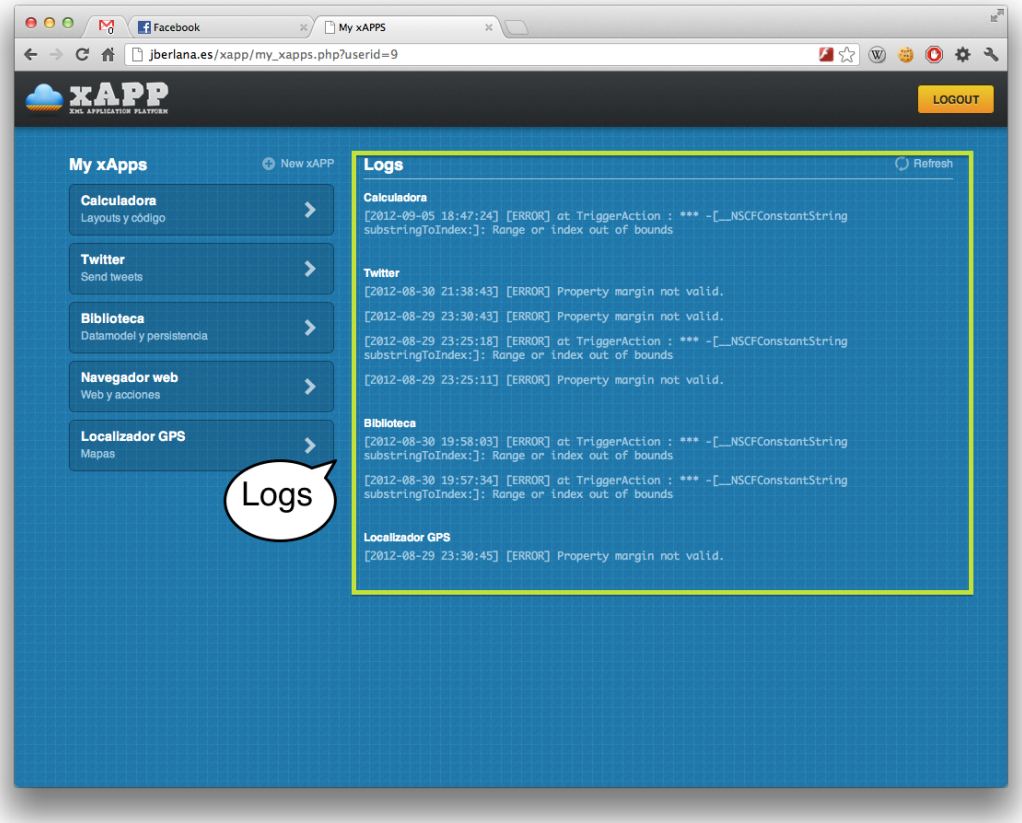

Figura 3.7: **Interfaz de inspección de logs -**

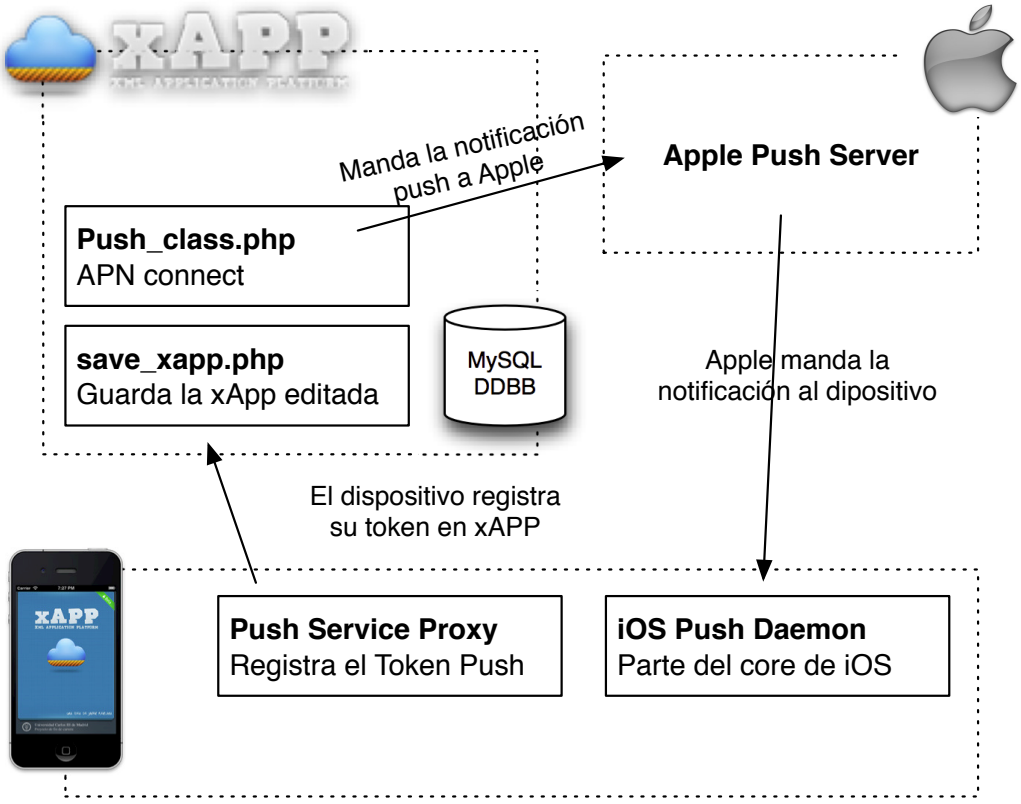

Figura 3.8: **Arquitectura de servicios Push** -

Figura 3.9: **Aviso de actualización push** -

3.1.2.4 API

Los servicios web de xAPP son servicios RESTful; es decir, cada mensaje HTTP contiene toda la información necesaria para comprender la petición. Como resultado, ni el cliente ni el servidor necesitan recordar ningún estado de las comunicaciones entre mensajes. De ésta forma el cliente se autentica con el server sin la necesidad de requerir de mecanismos para mantener el estado de la sesión.

xAPP expone tres llamadas:

- Login: Utilizada para que los clientes instalados en los servicios web se logeen contra el servidor.
- Log: El cliente móvil envía una traza al server cuando se produzca un error, esto permite realizar un debug básico de las aplicaciones en remoto.
- Get xApps: Una vez el usuario ha hecho login con ésta llamada, podrá descargarse todas las xApps creadas por él en del servidor.

Más adelante, en el apartado 'Implementación del entorno web',sección 4.2, se explicará con detenimiento los parámetros que reciben, las respuestas que producen y el funcionamiento interno de estos métodos.

3.1.3 Estructura de una app en xAPP

La estructura de un xApp viene definida por un esquema XML que es auto-validado a la hora de realizar el parseo. Se trata de una estructura simple pero eficaz para definir el árbol de vistas de la aplicación, el modelo de datos y el código.

Cómo se observa en el fragmento de código C.1, que define el esqueleto básico de una xApp, nos encontramos con un elemento principal `app` que puede contener en su interior de 1 a n `page`, la página raiz de la xApp siempre debe ser nombrada **home** y cada una de estas páginas definirán cada una de las vistas de la aplicación; además, la app podrá tener un elemento `datamodel` y `code` ambos opcionales, los cuales definirán el modelo de datos y el código interno de la xApp respectivamente.

Listing 3.1: Esqueleto de una xApp

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<app>
  <page id='home' cached='YES'>
    ...
  </page>
  <page id='other' cached='NO'>
    ...
  </page>
  ...
  <datamodel>
    ...
  </datamodel>
  <code>
    ...
  </code>
</app>
```

3.1.3.1 Las vistas

Las vistas de la xApp vienen definidas por elementos `page` contiene tres elementos principales que a su vez pueden contener sub-elementos:

- `objects` Declara objetos que se utilizarán dentro de la página.
- `header` que define el estilo y contenido de la barra de navegación. El header puede incluir una serie de propiedades para definir colores y fuentes, así como botones que se situarán en la parte derecha del navigatión bar. Se explicará con mayor profundidad cada uno de los elementos y propiedades que puede incluir

en el apéndice MANUAL DE PROGRAMACIÓN DE XAPPS. La figura 3.10 muestra la arquitectura de una página en xAPP.

- `content` que define el contenido de la página, puede tener infinitos elementos y a que generador de vistas maneja la posibilidad de scroll en caso de que el contenido supere el alto de la pantalla.

  El `content` de una xAPP está formado por `layout`, `sec` y `element` estos son layouts, secciones y elementos. Un layout define la forma en la que se alinean las vistas en su interior; por defecto, XAPP alineará todos los elementos en sucesión uno debajo del otro, haciendo uso de `<layout type='horizontal'>` conseguiremos que esta alineación se realice horizontalmente. Un layout, además, puede subdividirse en secciones que a su vez pueden contener otros layouts o elementos.

- `footer` elemento opcional, al igual que el `header` que define contenido fijo en la parte baja de la pantalla.

En el fragmento de código C.2 se puede ver cómo se estructuran estos elementos dentro de la página. Tanto `content` cómo `header` pueden incluir propiedades que definen el aspecto de los mismos, como puede ser un color de fondo o degradados.

Listing 3.2: Esqueleto de una página

```
<page id="home" cached="NO">
  <objects>
    book : null;
  </objects>

  <header>
    <property type="title">Library</property>
    <element id="barBut1" type="Add">
      <property type="alignment">right</property>
      ...
    </element>
  </header>

  <content>
    <property type="color">#cccccc</property>
    <element id="tableBooks" type="SimpleTable">
      <property type="height">100%</property>
      ...
    </element>
    ...
  </content>
```

```
22
23    <footer>
24    ...
25    </footer>
26  </page>
```

El `content` incluye, además, la sucesión de elementos de los que se componga la página, estos elementos dependen de la versión que se esté utilizando del framework. Cada uno de estos elementos define vistas de diferentes tipos, controladores, botones, labels de texto... El estado actual de los elementos disponibles y sus propiedades será detallado posteriormente en el capítulo Implementación, en la sección que define todos los componentes de la librería nativa.

La arquitectura general de un elemento se muestra en el fragmento de código C.3. Todo elemento tiene dos atributos obligatorios:

- `id`: Identificado único del elemento, podemos verlo cómo el nombre que se le da al objeto del tipo Element, y será el nombre con el que se le referencie desde otros objetos.
- `type`: Determina el tipo de elemento que estamos creando, viene dado por el nombre del módulo del elemento en xAPP; por ejemplo, SimpleButton, SimpleLabel o SimpleTextField. Dependiendo del tipo tendrá unas propiedades disponibles u otras.

Listing 3.3: Definición de un elemento

```
1  <element id='0' type='FooObject'>
2    <property type='text'>{object:book.title}</property>
3    <property type='title'>0</property>
4    <property type='action'>
5      current:concatText(0);
6      temp:editText({var:home.current.text});
7    </property>
8  </element>
```

Todo elemento tendrá de 1 a n `property`, y el contenido de una `property` puede ser de tres tipos diferentes.

- Descriptivas básicas: Definen propiedades del elemento con constantes, en el ejemplo `<property type='title'>0</property>`
- Descriptivas complejas: Definen propiedades del elemento con datos variables o dependientes de otro elemento de la xApp.

Las tenemos de dos tipos, las que cogen el valor de un objeto de la xApp: {`object:book.title`}. Donde `object` indica que el valor se encuentra en el modelo de datos y `book.title` que se trata de la columna de `title` de la tabla `book`.

Las que toman el valor de otro elemento de la xAPP por ejemplo:

{`var:home.current.text`} Donde `var` indica que el valor será tomado de otro elemento y `home.current.text` la sucesión: id de página, id de elemento y propiedad del elemento.

- El último tipo corresponde a las propiedades de tipo action, estas propiedades se pueden dar en botones o elementos que quieran lanzar una determinada acción cuando se produzca un determinado evento.

  La estructura de las acciones es la siguiente:

  `[id de la página]<id del elemento>:<nombre del método>([parámetros]);`

  Los parámetros vienen definidos por la estructura vista en las propiedades descriptivas complejas del punto anterior.

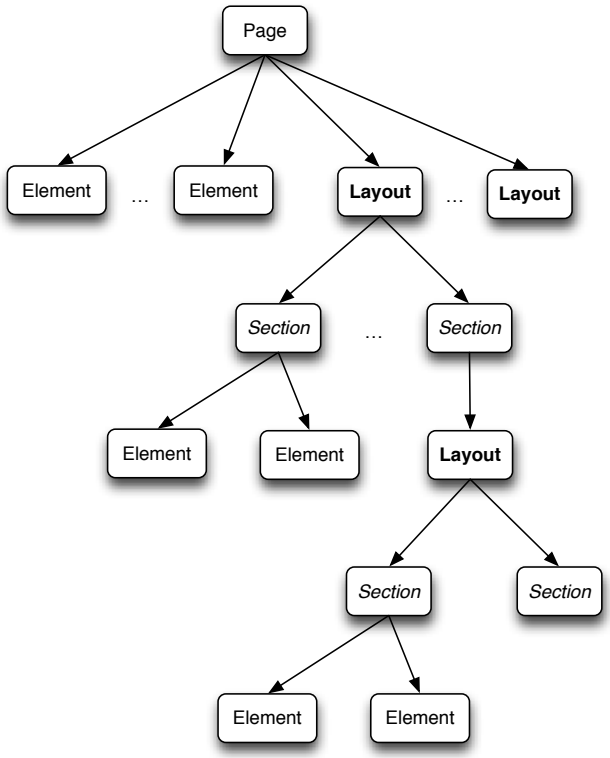

Figura 3.10: **Árbol de elementos de una página** -

3.1.3.2 El modelo de datos

Aquellas xApps que requieran de persistencia de datos o necesiten prepoblar una serie de páginas iguales con diferentes datos sin duplicar la vista, es decir hacer uso del patrón Modelo-Vista-Controlador, utilizando una estructura maestro-detalle pueden hacer uso de esta característica.

Un ejemplo de data model es el siguiente, que se muestra en el fragmento de código C.4

Listing 3.4: Ejemplo de datamodel

```
<datamodel>
  <book>
    <id>1</id>
    <title>El nombre del viento</title>
    ...
  </book>
  <book>
    <id>2</id>
    <title>El temor de un hombre sabio</title>
    ...
  </book>
  ...
</datamodel>
```

El datamodel sirve tanto para definir el esquema del modelo de datos, como para poblarlo; si únicamente se quiere definir el esquema basta con crear las tablas con los elementos vacíos, si queremos poblarlos crearemos tantos elementos como filas queramos insertar en nuestra tabla. En el ejemplo del fragmento de código C.4 se ha creado la tabla `book` que tiene las columnas `id` y `title` y se han insertado dos filas de libros diferentes.

xAPP permite mantener n tablas con m columnas por xAPP, siendo el único requisito que todas las tablas con el mismo nombre mantangan el mismo número de columnas con mismo nombre, es decir mantengan un esquema coherente. En otro caso se producirá un error.

3.1.3.3 El código

En cualquier programa informático la información se recupera, se almacena y se transforma. Hemos visto cómo obtenerla a través de servicios web, cómo almacenarla haciendo uso del modelo de datos, y ahora veremos cómo podemos transformar esta información mediante código Javascript.

El código se incluye en la sección `code` de la xApp, el código permitido son funciones Javascript, como la del siguiente ejemplo:

Listing 3.5: Ejemplo de código

```
<code>

function getResult(temp, opp, current){
    var result = eval(temp + opp + current);
    to_xapp('home.current:setText('+result+')');
}

</code>
```

En el ejemplo se define una función en javascript que recibe tres valores por parámetro y devuelve un resultado por medio de la función interna `to_xapp()`. Para llamar a ésta función desde un elemento `action` de la xApp se utiliza la nomenclatura:

```
app:<nombre del método>([parámetros]);
```

Por ejemplo, en el caso del código definido en C.5 la llamada se realizará de la siguiente forma:

```
app:getResult(4,'+',3);
```

En el apartado para código de la xAPP podemos definir variables locales, funciones recursivas, en definitiva, cualquier código javascript válido. Como hemos visto en la sección 4.1.4 JS Core, xAPP corren una instancia del interprete de Javascript del navegador nativo de la plataforma que se está utilizando, y ejecuta en él el código introducido. Siendo el punto de entrada la llamada por medio de `app:<método>();` y la salida la ejecutada en la función `to_xapp()`.

Capítulo 4

Implementación

4.1 Librería nativa para iOS

En este capítulo se detalla cómo se ha implementado cada una de las parte de las que se compone el Framework, se empezará describiendo qué y cómo se han desarrollado cada uno de los componentes de los que se compone el Framework, pasando a mostrar algunas aplicaciones de los mismos para desarrollar ejempos funcionales.

El capítulo termina describiendo la implementación de los servicios en la nube y de cómo se ha desarrollado el API web para acceder a esta funcionalidad desde la libreria de xAPP para iOS.

4.1.1 Parser de XML

Siguiendo lo descrito en el diseño el parser implementado es un parser de tipo SAX, que lee el documento XML secuencialmente de principio a fin. Implementa el protocolo definido por `NSXMLParserDelegate` este protocolo genera los 3 eventos clásicos de un parser SAX:

- Al inicio de un elemento.
- Por cada caracter dentro del elemento.
- Al finalizar el elemento.

Todo documento XML comienza con una etiqueta de inicio y termina con una etiqueta final a juego, o consta solo de una etiqueta de elemento vacío. Los caracteres entre el inicio y final de las tags, si los hay, son el contenido del elemento, y puede contener marcas, incluyendo otros elementos, que se llaman elementos secundarios. Un ejemplo de un elemento es `<property type='height'>10</property>`.

Las etiquetas pueden contener atributos, los atributos van dentro los símbolos `<` y `>`; en el ejemplo anterior `<property type='height'>10</property>` type sería el atributo asociado a la etiqueta property y 10 el valor del elemento.

En el fragmento de código 4.1 se muestran los objetos que mantiene en memoria el parser durante el proceso de lectura del XML.

Listing 4.1: Atributos del parser

```
1  @property (nonatomic, retain) NSMutableString *currentString;
2  @property (nonatomic) BOOL storingCharacters;
3
4  @property (nonatomic, retain) PageContainer *currentPage;
5  @property (nonatomic, retain) Layout *currentlayout;
6  @property (nonatomic, retain) Section *currentSection;
7  @property (nonatomic, retain) Element *currentElement;
8
9  @property (nonatomic, retain) NSMutableDictionary *pages;
10 @property (nonatomic, retain) NSMutableArray *nestedLayoutsHeap;
11 @property (nonatomic, retain) NSMutableArray *sectionsHistoryHeap;
12 @property (nonatomic, retain) NSString *currentProperty;
13 @property (nonatomic, retain) NSString *code;
14
15 @property (nonatomic, retain) NSMutableArray *dataModelHeap;
16 @property (nonatomic, retain) NSMutableArray *dataModelNodesHeap;
```

```
17 @property (nonatomic, retain) NSMutableDictionary *currentDataModel;
18 @property (nonatomic, retain) NSString *currentRoot;
```

4.1.2 SQL Core y Generador de modelos de datos

El siguiente fragmento de código muestra las cabeceras de la clase `SqlCore` que es la encargada de manejar la persistecia de datos en los dipositivos.

Una vez termina el parser de leer el modelo de datos, el generador de vistas recorrerá la estructura de datos generada creando un script de creación SQL, si la base de datos no estuviese creada. Una vez se crea la vase de datos se generan las sentencias de inserción correpondientes para poblarlas con los datos recibidos del código XML.

Listing 4.2: Interfaz de SqlCore

```
1  #import <Foundation/Foundation.h>
2  #import "FMDatabase.h"
3
4  @interface SqlCore : NSObject
5
6  @property (nonatomic, retain) FMDatabase *db;
7  @property (nonatomic, retain) NSDictionary *schema;
8
9  - (BOOL)setDBwithPath:(NSString *)name;
10 - (BOOL)createDbFromDataModel:(NSDictionary *) models;
11 - (BOOL)populateDbWith:(NSDictionary *)models;
12
13 - (BOOL)syncDBwith:(NSDictionary *)dict;
14 - (BOOL)storeInDbObject:(NSDictionary *)dict;
15
16 - (NSArray *)selectRowsFor:(NSDictionary *)params withName:(NSString *)
       tableName;
17 - (NSDictionary *)getSchemaOfTable:(NSString *)tableName;
18 - (NSDictionary *)selectRowId:(NSString *)rowId withName:(NSString *)tableName
       ;
19 - (NSDictionary *)selectRow:(NSString *)theQuery withName:(NSString *)
       tableName;
20 - (NSString *)selectValue:(NSString *)value For:(NSDictionary *)params
       withName:(NSString *)tableName;
21 - (NSString *)selectValue:(NSString *)value withQuery:(NSString *)theQuery
       withName:(NSString *)tableName;
22 - (void)deleteRowInTable:(NSString *)tableName withID:(int)xappid;
```

```
23
24  - (void)createXappDB;
25
26  @end
```

Como se puede ver, ésta clase expone los métodos necesarios para obtener e insertar datos en la base de datos desde cualquier punto de ejecución de la aplicación.

4.1.3 Comunicación con webservices

Este módulo se encarga de enlazar el cliente nativo con los componentes de xAPP en la nube.

En la librería para iOS, este módulo consta de un cliente de AFNetworking, `XappServerClient.m`, que maneja los datos de acceso del usuario, controlando datos coomo el user id o el fingerprint utilizado para acreditar a los usuarios y gestionar sus credenciales. Esta clase es un Singleton, manteniendo de esta manera una única instancia de si mismo durante la ejecución del cliente nativo.

Son varias las clases que interactuan con este módulo:

- `User.m`: Para hacer login y controlar las credenciales de acceso del usuario.
- `XAPP.m`: Para descargar una determinada xApp, o todas las del usuario.
- `ServerLogs.m`: Para mandar logs al servidor en caso de qeu se produzcan errores en la carga de las xApps.

4.1.4 JS Core

Este módulo es el encargado de correr el código de cada una de las xApps. Para ello se crea una instancia de UIWebView en la clase Singleton `JSManager.m`. Creando una instancia de un WebView y corriendo en este nuestro script conseguimos utilizar el interprete Javascript de Safari, `javaScriptCore`, consiguiendo el mejor performance posible en la ejecución de código javascript.

Esta clase llamada desde `PageViewController.m` cuando alguno de sus elementos necesiten ejecutar alguna función propia de la xApp, se llamará a la función y se devolverá el resultado.

Estas llamadas se realizan de la siguiente manera:

```
1
2  // JSManager.m
3  - (void) callFunction:(NSString *)query
```

```
4  {
5      [loader stringByEvaluatingJavaScriptFromString:query];
6  }
7
8  // Loader.m
9  - (void)webView:(UIWebView *)webView runJavaScriptAlertPanelWithMessage:(
       NSString *)message initiatedByFrame:(id)frame
10 {
11     NSLog(@"Message: %@", message);
12     [self.managerDelegate loaderDidReceiveResponse:message];
13 }
14
15
16 // AppDelegate.m
17 - (void)loaderDidReceiveResponse:(NSString *)response
18 {
19     NSLog(@"%@", response);
20
21     int a = [response rangeOfString:@"."].location;
22     int b = [response rangeOfString:@":"].location;
23     int c = [response rangeOfString:@"("].location;
24
25     NSString *pageId = [response substringToIndex:a];
26
27     NSRange rango = NSMakeRange(a+1, b-a-1);
28     NSString *elementId = [response substringWithRange:rango];
29
30     rango = NSMakeRange(b+1, c-b-1);
31     NSString *action = [response substringWithRange:rango];
32
33     rango = NSMakeRange(c+1, response.length-c-2);
34     NSArray *args = [[response substringWithRange:rango]
           componentsSeparatedByString:@","];
35
36     PageViewController *viewController = [self.pagesCached objectForKey:pageId
           ];
37     ElementView *element = [viewController.contentControllers objectForKey:
           elementId];
38
39     [element becomeFirstResponder];
40
```

```
41      SEL selector = NSSelectorFromString([NSString stringWithFormat:@"%@:",
            action]);
42
43      @try {
44          [element performSelector:selector withObject:args];
45      }
46      @catch (NSException *exception) {
47          NSLog(@"[ERROR] '%@' not response for '%@'",elementId,action);
48      }
49
50  }
```

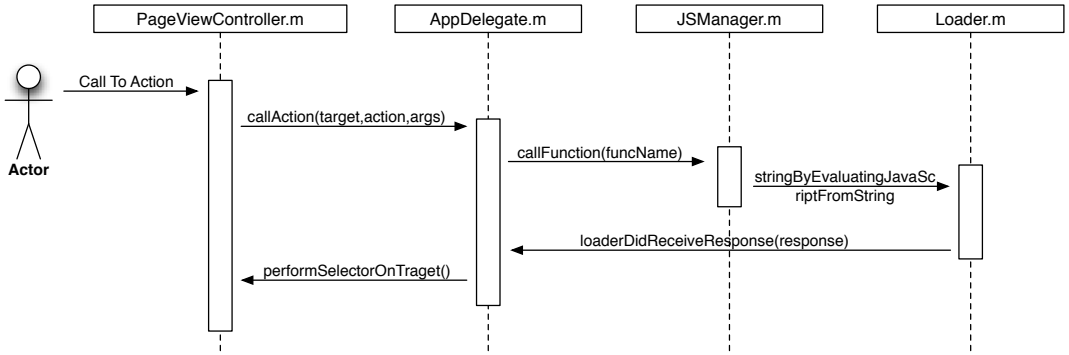

Figura 4.1: **Diagrama de flujo de carga de javascript** -

En la figura 4.1 se observa la sucesión de llamadas que se realizan desde que es lanzado el evento hasta que se obtiene la respuesta del `JSManager`. Todas las operaciones se realizan en segundo plano, con el fin de no bloquear la interacción del usuario con la aplicación.

4.1.5 Componentes

xAPP es un Framework pensado para ser modular y fácilmente extensible. Con esto pretendo que cualquier persona sea capaz de diseñar elementos y controladores y pueda añadirlos al Framework sin necesidad de tocar ninguna otra clase. Es decir, basta con crear una clase (.m y .h en caso de la librería para iOS) e insertarla en la carpeta `Elements` del proyecto.

Para ello todos los elementos nuevos deben extender de `ElementView` e implementar el protocolo `ElementProtocol`; haciendo esto podremos añadir elementos a nuestras xApps indicando en el tipo de los elementos incluidos en el XML el nombre que le hemos dado a las clases que estamos creando. Por ejemplo, si hemos creado `MyNewController.m` crearemos un elemento de este tipo con
`<element id=10 type=MyNewController>`

```
1 - (id)initWithProperties:(NSDictionary *)theProperties
2               andParent:(PageViewController *)theParent
3                andFrame:(CGRect)theFrame
```

Todos los controladores deben implementar el constructor definido en el padre, quien recibe las propiedades del elemento además del frame asignado por su `PageViewController`. Al tener todo elemento una referencia a `PageViewController` conseguimos tener un contexto de manera que se pueda acceder a propiedades o valores de otros elementos con los que comparte página.

4.1.5.1 Botón

Elemento básico de cualquier aplicación móvil, ya que es uno de los elementos con el que el usuario puede interactuar para desencadenar una determinada acción.

xAPP permite crear botones con un estilo personalizado en muy pocas líneas, dando

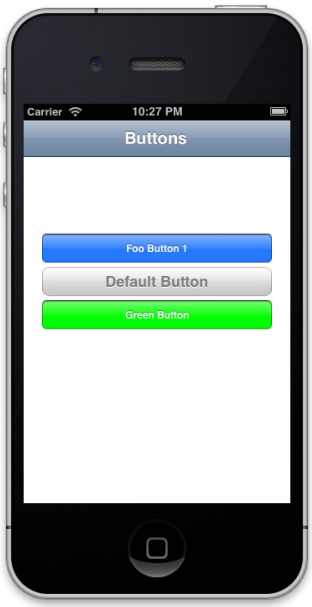

Figura 4.2: **Ejemplo de botón** -

resultados rápidos y mucho más visuales que los botones nativos, pudiendo aplicar colores por código RGB para personalizar su color, opción que no está disponible en el SDK nativo.

En iOS el botón de xAPP utiliza internamente un `UISegmentedControl` con un único item, y la propiedad `momentary` activada. De ésta manera se consigue un efecto glass, o cristal, sobre el botón que con `UIButton` no se podría conseguir, a menos que utilicemos imágenes como fondo para el botón.

Las propiedades que admite un botón son las siguientes:

- **title**: Define el texto que aparecerá en el botón.
- **color**: Define el color del botón, debe ser un código RGB.
- **margin**: Pixeles en blanco que dejará el botón a la derecha e izquierda.
- **height**: Altura del botón.
- **action**: Acción que se realizará cuando el botón reciba un evento de tipo tap_inside.

Listing 4.3: Código para un botón

```
<element id="button1" type="SimpleButton">
  <property type="margin">20</property>
  <property type="height">40</property>
  <property type="title">Foo Button</property>
  <property type="color">#4099FF</property>
  <property type="action">
    home:foo(2);
  </property>
</element>
```

El botón incluido en xAPP además permite realizar tareas como mandar un email o twitear, simplemente con definir la acción del tipo:

- `tweet()`: Para lanzar un Tweet.
- `mail()`: Para mandar un email.

Ambas reciben por parámetro el texto que se quiere enviar.

4.1.5.2 Label

Un label es un texto corto, en xAPP de menos de una linea, comúnmente usado para describir otro elemento o mostrar información en pantalla. Las Labels de xAPP utilizan, en el caso de la librería nativa para iOS, el elemento del sistema UILabel.

Las propiedades que admite un botón son las siguientes:

- **text**: Define el texto que aparecerá en el label.
- **color**: Define el color del label, debe ser un código RGB.
- **alignment**: Alineación del label en la página, (center, left o right).
- **font**: Tipo de fuente que se utilizará.
- **size**: Tamaño de la fuente.
- **alpha**: De 0 a 1, define la opacidad del label

Este control adicionalmente puede recibir como property cualquier atributo válido del elemento `UILabel` de iOS. Esto es así por el modo en el que ha sido implementado el set de los properties no normalizados. Observese el fragmento de código 4.7.

Listing 4.4: Propiedades dinámicas

```
for (uint i=0; i<super.properties.count; i++)
{
  NSString *key = [super.properties.allKeys objectAtIndex:i];
  NSString *val = [super.properties objectForKey:key];
  SEL setter = NSSelectorFromString([NSString stringWithFormat:@"set%@:", [key
       capitalizedString]]);
  [self performSelector:setter withObject:val];
}
```

Lo que hace es recorrer las propiedades definidas que no han sido ya utilizadas creando un `@selector`[1] con cada una de ellas que es lanzado usando como target la propia label.

Listing 4.5: Código para label

[1]En Objective-C, un selector tiene dos significados. Puede ser utilizado para referirse simplemente a el nombre de un método cuando se utiliza en un mensaje a un objeto. Pero también se refiere al identificador único que reemplaza el nombre cuando el código fuente se compila. Selectores compilados son de tipo SEL. Todos los métodos con el mismo nombre tienen el mismo selector. Puede utilizar un selector para llamar a un método en un objeto, lo que proporciona la base para la implementación del patrón de diseño de action target en Cocoa.

```
<element id="myLabel" type="SimpleLabel">
  <property type="text">Hello World!</property>
  <property type="color">#333399</property>
  <property type="alignment">center</property>
  <property type="font">Helvetica-Bold</property>
  <property type="size">50</property>
</element>
```

Figura 4.3: **Ejemplo de label** -

4.1.5.3 Campo de texto

Los campos de texto son muy utilizados en las aplicaciónes móviles y uno de los controles que no podía faltar en esta primera versión de xAPP. Es el elemento que utilizará el usuario para introducir información en la aplicación en forma de texto.

La librería de xAPP para iOS utiliza internamente el componente UITextField y admite todas las propiedades nativas del componente, cómo son:

- **text**: Define el texto que aparecerá en el TextField.
- **textalignment**: Alineación del texto en el TextField, (center, left o right).
- **font**: Tipo de fuente que se utilizará.
- **size**: Tamaño de la fuente.
- **mapping**: Permite mapear el textfield con un objeto de la página.

Figura 4.4: **Ejemplo de textfield** -

Es importante destacar la propiedad `mapping`, que nos permite mapear un TextField a un objeto definido en la página, de manera que si éste objeto pertenece al modelo de

datos y está marcado como persistente, nos permitirá modificar su valor el la DB una vez el usuario termine de escribir.

Listing 4.6: Código para SimpleTextField

```
<element id="text2" type="SimpleTextField">
  <property type="placeholder">Foo field 2</property>
  <property type="textalignment">center</property>
  <property type="font">Helvetica-Bold</property>
  <property type="size">30</property>
  <property type="mapping">{object:book.author}</property>
</element>
```

En el fragmento de código anterior se mapea el valor del SimpleTextField con el atributo `author` del objeto `book` definido en esa misma página. Esto se consigue gracias una de las características de Objective-C los selectores, en el siguiente fragmento de código se muestra cómo:

Listing 4.7: Propiedades dinámicas

```
-(void) setMapping:(NSString *)property
{
    if ([property hasPrefix:@"{object:"])
    {
        NSString *query = [property substringFromIndex: [property rangeOfString
            :@":"].location+1];
        query = [query substringToIndex:query.length-1];
        NSArray *tokens = [query componentsSeparatedByString:@"."];
        self.mapedValue = tokens;
    }

    [self.theTextField addTarget:self action:@selector(updateObject)
        forControlEvents:UIControlEventEditingDidEnd];
}

- (void)updateObject
{
    if (self.theTextField.text.length > 0)
    {
        NSString *prop = [[self.mapedValue lastObject]
            stringByTrimmingCharactersInSet:[NSCharacterSet
            whitespaceCharacterSet]];
```

```
19          NSString *key = [[self.mapedValue objectAtIndex:0]
                stringByTrimmingCharactersInSet:[NSCharacterSet
                whitespaceCharacterSet]];
20
21          [[self.parent.page.objects objectForKey:key] setObject:self.
                theTextField.text forKey:prop];
22      }
23  }
```

El primer método es llamado cuando el `View Drawer` se encuentra con una propiedad del tipo `mapping` parseando en el objeto, almacena el objeto y la propiedad parseada, en la linea 11 establece la acción `updateObject` para el evento `UIControlEventEditingDidEnd`. De esta manera, cuando el usuario termine de introducir texto en el campo se lanzará el segundo método, el cual se encargará de cambiar el valor del atributo con el texto introducido.

También es importante destacar que el SimpleTextField es capaz de recibir acciones; es decir, puede ser invocado por otros elemento con el fin de modificar su estado. Estas acciones son:

- **concatText**: Concatena texto al final del texto actual.
- **setText**: Cambia el texto completo en el textfield.

Por ejemplo:

Listing 4.8: Código para SimpleTextField

```
1  <element id="A" type="SimpleTextField">
2    <property type="textAlignment">right</property>
3    <property type="size">50</property>
4    <property type="text">0</property>
5  </element>
6
7  <element id="B" type="SimpleButton">
8    <property type="title">B</property>
9    <property type="action">current:concatText(0)</property>
10 </element>
```

En este fragmento de código tenemos un botón B que al ser pulsado concatena un 0 al final del texto en A.

4.1.5.4 Imágenes

Las imágenes son un elemento básico de cualquier aplicación móvil que no podían faltar en xAPP; actualmente el entorno web no permite la subida de ficheros por lo que las imágenes se cargan desde URLS de internet.

Internamente en iOS se utiliza el componente UIImageView para pintar la imagen que se obtiene de la URL en pantalla. El uso es muy sencillo, simplemente basta con rellenar los atributos de la siguiente lista, aunque también se pueden incluir otrar propiedades que acepte UIImageView.

- **image**: Para indicar la URL de la imágen a cargar.
- **width**: Ancho de la imagen.
- **height**: Alto de la imagen.
- **align**: Alineación de la imagen en la página, (center, left o right).

Figura 4.5: **Ejemplo de imagen en una xApp** -

4.1.5.5 Mapas

Elemento usado para geoposicionar al usuario o mostrar un punto de interés, internamente hace uso de los mapas de Google, a través del componente `MKMapView`. Las propiedades que admite son las siguientes:

- **height**: Altura que ocupará el mapa en la página actual.
- **border**: YES o NO si queremos que el mapa se muestre con borde.

Pero además puede recibir dos tipos de acciones:

- **goTo([destino],[origen])**: Mostrara la ruta entre los dos puntos.
- **lookFor([dirección])**: Realizará una búsqueda de la dirección escrita y mostrará en el mapa la primera posición devuelta por GOOGLE.

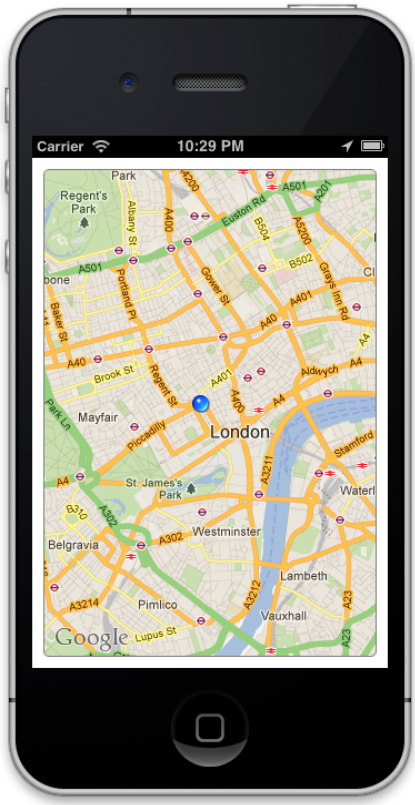

Figura 4.6: **Ejemplo de mapView** -

xAPP simplifica notablemente el uso de mapas en una aplicación móvil, en pocas lineas conseguimos lo que en desarrollo nativo llevaría unos miles de lineas y el uso

de numerosas clases y librerías. iOS, por defecto, no ofrece la posibilidad de hacer búsquedas ni pintar rutas sobre los mapas por lo que esta funcionalidad ha sido desarrollada desde cero.

Para poder hacer búsquedas se ha desarrollado un modulo `geocoder` haciendo uso de las APIs sobre geolocalización que ofrece Google Maps; éste modulo basicamente crea un request en JSON que envía al API de geocoding de Google Maps, recibe la respuesta JSON, la parsea y posiciona un pin en el lugar indicado por Google Maps, como se puede observar en el siguente diagrama de secuencia.

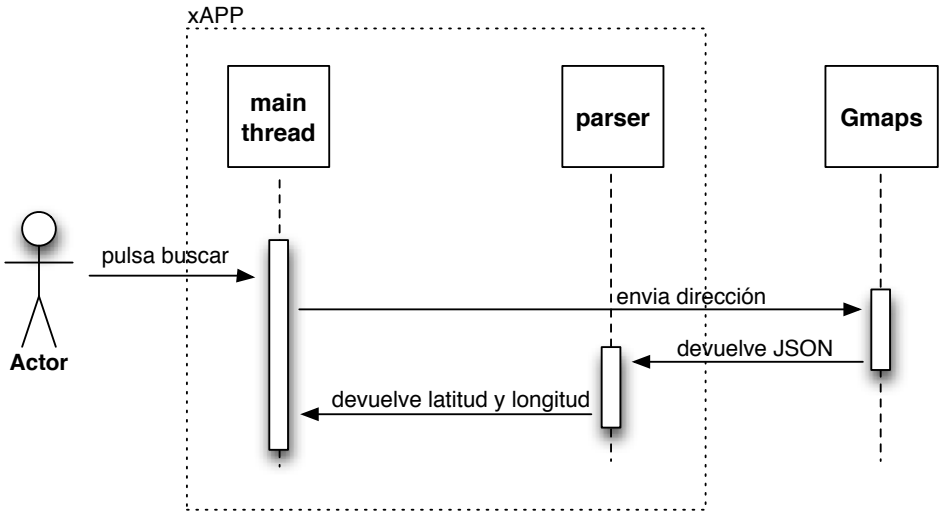

Figura 4.7: **Diagrama de secuencia del geocoder** -

De manera similar funciona el cálculo de rutas, se envía a Google Maps el punto de origen y destino y se recibe un JSON con cada uno de los puntos de la ruta. Haciendo uso de CoreGraphics se crea un `poliline` que se coloca sobre el mapa en forma de capa indicando al usuario la dirección que tendrá que recorrer.

4.1.5.6 Webview

SimpleWebView es el módulo incluido en xAPP que se encarga de colocar una vista web directamente dentro de tus xApps, está basado en el controlador UIWebView que ofrece el UIKit de APPLE.

Figura 4.8: **Ejemplo de textfield** -

Los parámetros que acepta este componente son los siguientes:

- **url**: La URL de la web a cargar.
- **height**: Altura que ocupará el mapa en la página actual.
- **border**: YES o NO si queremos que el mapa se muestre con borde.

Y además puede recibir los siguientes tipos de acciones:

- **goTo([url])**: Cargará la página indicada en el webView.
- **goBack()**: Típica acción atrás de un navegador web.
- **goForward()**: Volverá hacia la pagina cargada antes de pulsar atrás.

4.1.5.7 Tablas

En toda aplicación móvil es necesario mostrar un listado de elementos, por lo que en xAPP no podía faltar ésta funcionalidad.

Al contrario del resto de elementos en xAPP, que anteriormente hemos visto que extendian de `ElementView`, las tablas extienden de `UITableViewController` aunque como el resto de elementos implementa el protocolo `ElementProtocol` para exponer sus propiedades y poder acceder a las del resto de elementos.

SimpleTableView necesita de un modelo de datos definido en la xApp para poder funcionar ya que lo utilizará cómo `datasource`. El siguiente fragmento de código muestra cómo se define un tableview.

Listing 4.9: Código para tablas

```
<element id="tableBooks" type="SimpleTable">
  <property type="height">100%</property>
  <property type="width">100%</property>
  <property type="table">book</property>
  <property type="title">title</property>
  <property type="subtitle">author</property>
  <property type="action">home:pushPage(detail)</property>
</element>
```

Los parámetros que puede recibir son los siguientes:

- **height**: Porcentaje de altura que ocupará en la tabla en la pantalla.
- **width**: Porcentaje de anchura que ocupará en la tabla en la pantalla.
- **table**: Nombre del tabla del modelo de datos que se utilizará para rellenar la tabla, en el ejemplo podemos ver como se utilizará la tabla llamada `book`.
- **title**: Nombre del atributo de la tabla seleccionada que se utilizará cómo titulo principal para celda, en el caso del ejemplo corresponde a `title`.
- **subtitle**: Nombre del atributo de la tabla seleccionada que se utilizará cómo subtitulo para celda, en el caso del ejemplo corresponde a `author`.
- **action**: Acción que se lanzará cuando el usuario haga tap sobre una determinada celda. Además la tabla pasará como parámetro el objeto sobre el que se ha hecho tab, que la siguiente página recibirá con:

```
<objects>
  book : {prev.book};
</objects>
```

- **delete**: True en caso de que queramos que el usuario pueda eliminar celdas de la tabla. Al eliminarse una celda se eliminará también la fila relacionada el modelo de datos.

Figura 4.9: **Ejemplo de tabla** -

4.2 Entorno Web

xAPP tiene un fuerte componente web, ya que toda la gestión y creación de xApps se realiza desde la nube.

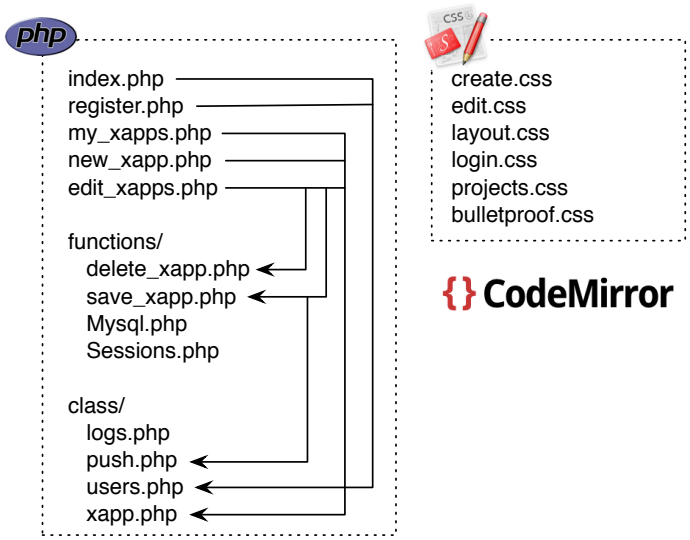

Figura 4.10: **Arquitectura web** -

Todo el desarrollo ha sido realizado en PHP 5 con orientación a objetos, HTML 5 para el frontend web, de manera que la web sea accesible desde cualquier dispositivo móvil, permitiendo edición y ejecución sin necesidad de un PC. En la figura 4.10 se pueden ver cada unos de los ficheros y clases de las que hace uso xAPP. Para la gestion de las bases de datos se utiliza MySQL.

La figura 4.11 muestra el modelo relacional bajo el que se ha construido la base de datos que soporta todo el Framework. Es un modelo muy sencillo que permite mantener múltiples xApps por usuario, instalaciones de xApps aunque el usuario no sea el creador (introduciendo xApps públicas, aunque no implementado) y almacenamiento de logs. Todas las consultas a la base de datos han sido realizadas usando técnicas seguras de acceso a DB para evitar ataques de SQL Injection.

4.2.1 Registro y login de usuarios

Permite al usuario crear una cuenta en xAPP y hacer login. Para el registro el usuario únicamente necesita introducir su email y una contraseña; posteriormente, cuando haga login en el móvil el sistema enlazará el identificador único de dispositivo con la cuenta del usuario para poder enviar notificaciones push cuando realice actualizaciones en sus xApps.

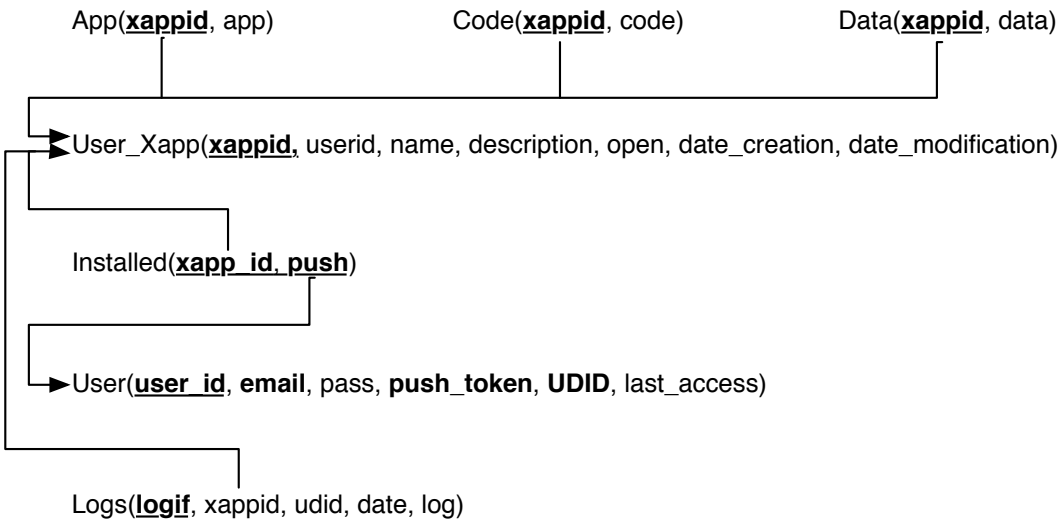

Figura 4.11: **Modelo Relacional de la BBDD** -

El sistema no almacena contraseñas en claro del usuario, sino un hash Sha1 de la contraseña, de manera que en caso de verse vulnerada la seguridad del servidor la privacidad de los usuarios no se vea afectada.

Una vez el usuario ha hecho login las credenciales se mantienen haciendo uso de sesiones, en la sesión se almacenará por un lado el user_id del usuario y por otro un fingerprint. Este fingerprint es utilizado para evitar el robo de sesiones o que un usuario pueda acceder a las xApps de otro; se trata de un hash md5 con salt de la el user_id, la IP y el useragent del usuario.

En cada cambio de página se comprueba que la sesión y el fingerprint sean válidos, evitando de esta manera posibles ataques de seguridad.

4.2.2 Visualización de logs

Desde la página principal, una vez el usuario ha realizado login satisfactoriamente, se muestra al usuario una vista con los logs que se han producido en cada una de sus xApps ordenados por fecha y agrupados por aplicación.

Los logs se reciben mediante el API de logs, que se comentará más adelante, y en esta vista son extraidos de la base de datos con simples consultas sql.

La figura 4.13 muestra el listado de logs junto a la lista de aplicaciones del usuario.

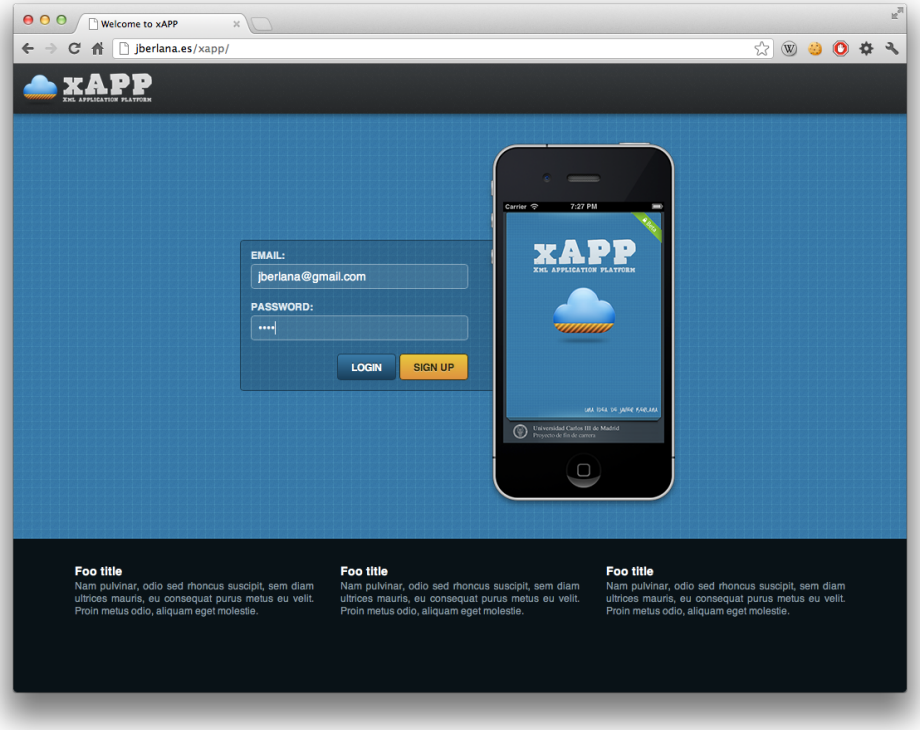

Figura 4.12: **Pantalla de login** -

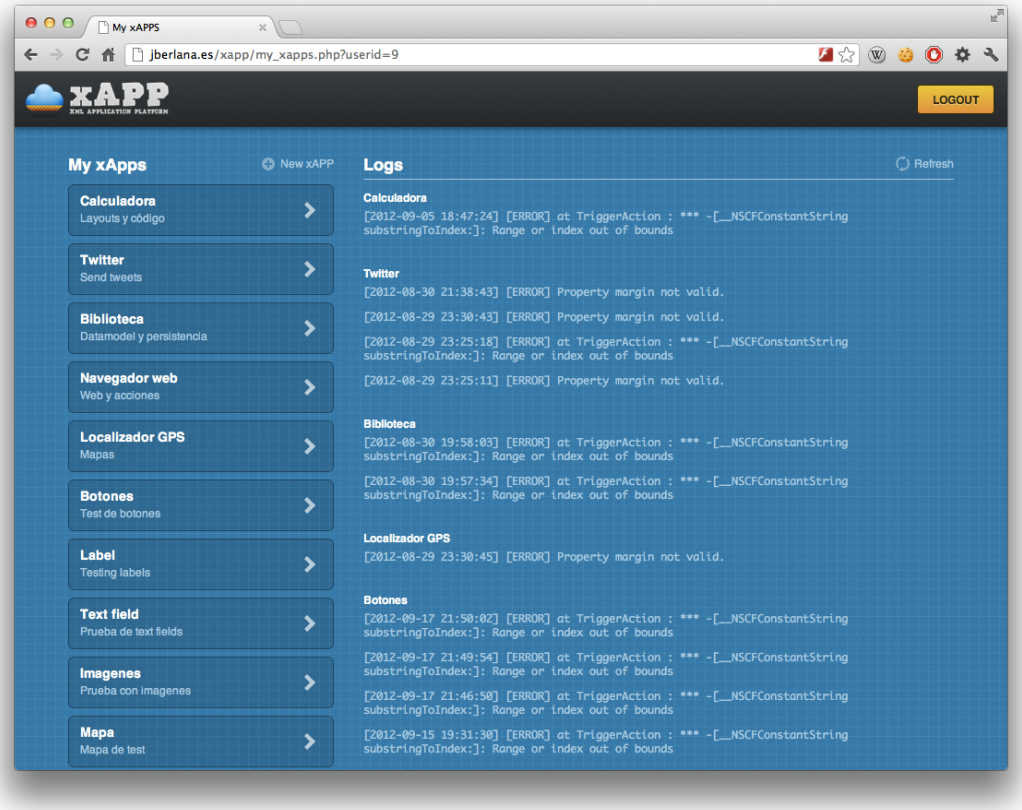

Figura 4.13: **xApps del usuario y logs -**

4.2.3 Creación y edición de xAPPs

El usuario puede editar sus xApps desde la web, como se puede observar en la figura 4.14, para poder tener soporte de coloreado de código xAPP hace uso de la librería Codemirror escrita en javascript.

Al usuario se le muestran 3 pestañas: app, datamodel y código, correspondientes a toda la estructura de vistas de la aplicación, al modelo de datos y a las funciones javascript de las que hace uso la xAPP respectivamente.

El código de cada una de estas secciones se extrae de la base de datos mediante consultas SQL a las tablas `app`, `data` y `code` respectivamente.

Una vez el usuario pulsa en guardar, si se han producido cambios en la aplicación, se enviará una notificación PUSH a todos los dispositivos que tengan instalada la xApp modificada para que puedan actualizarse.

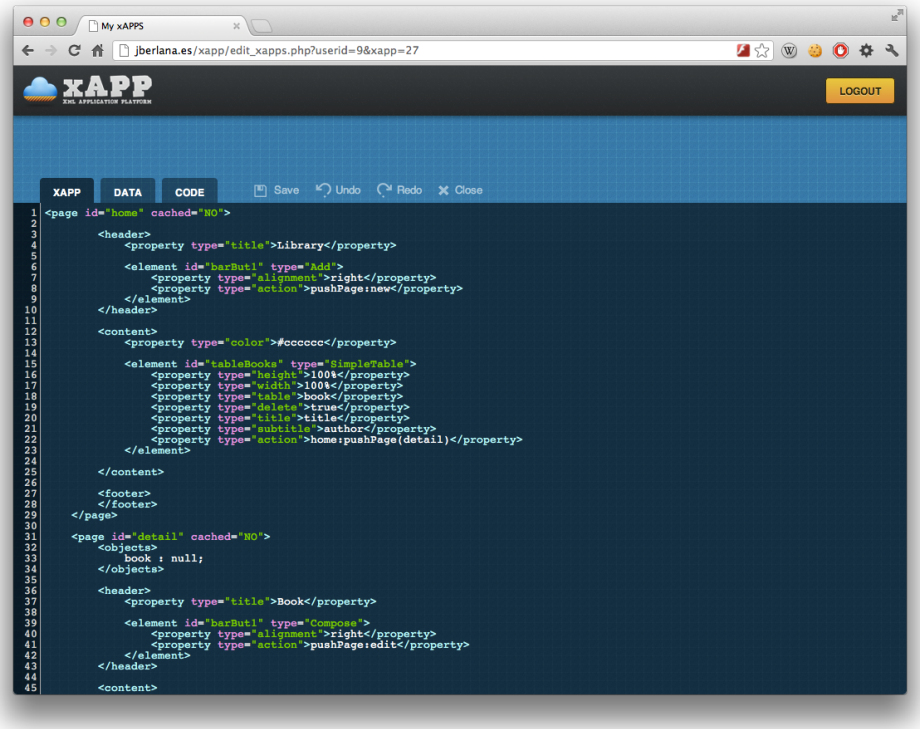

Figura 4.14: **Editor de código** -

4.3 Webservices

Los web services permiten comunicar a los dispositivos móviles con el sistema de back-end en la nube, donde se almacena el código de las xApps y las credenciales de los usuarios.

Los servicios web que se han construido para ésta versión de xAPP son los cuatro que se comentan a continuación. Han sido desarrollado haciendo uso de la tecnología RESTful y todas las respuestas son en JSON aligerando el tamaño de los datos transferidos en un 30% con respecto al XML.

4.3.1 Servicios Push

En xAPP se ha implementado un sistema de notificaciones Push como sistema de aviso de actualizaciones, de forma que siempre que se edite el código de una xApp todos los dispositivos que tengan esa xAPP instalada reciban una notificación Push para poder actualizarse.

En la figura 3.8 se muestra la arquitectura del sistema de notificaiones Push. El dispositivo registra a través de los web services su token push al realizar el login, posteriormente cuando una xApp sea editada se utilizará este token para manda una petición de notificación al servicio APNS de Apple, que será el encargado de notificar a los dispositivos implicados.

4.3.2 Login

Es el servicio encargado de validar las credenciales de un usuario de xAPP desde un terminal móvil.

`/api/login.php` recibe por parámetro los siguientes datos:

- **email**: El email del usuario que está haciendo login, dato introducido por pantalla.
- **password**: La contraseña del usuario en claro (cifrada en el canal HTTPS), dato introducido por pantalla.
- **udid**: Identificado único del dispositivo, se obtiene por medio de funciones del API de iOS.
- **push**: Token Push proporcionado por Apple para este dispositivo, se obtiene por medio de funciones del API de iOS.

Dado que se usa un servicio RESTful, una vez sean validadas las credenciales del usuario hay que proporcionar un sistema para autenticar el resto de llamadas al servicio web, ya que se carece de sesiones.

Para ello con el resto de peticiones al servicio web se enviará el user_id del usuario registrado junto con un fingerprint que consiste en un sha1 con los siguientes datos:

- email
- user_id
- sha1(password)

4.3.3 Descarga de xAPPs

Servicio disponible en la URL `api/getXapps.php` que permite obtener todas las xApps del usuario o solamente una xApp en caso de que se envíe el identificador de la xApp.

Recibe los siguientes parámetros:

- **userid** (Obligatorio): Identificador del usuario.
- **fingerprint** (Obligatorio): Token generado al realizar el login.
- **push** (Obligatorio): Token Push del usuario.
- **xappid** (Opcional): En caso de enviar este id solo se recibirá el código de la xApp solicitada, en caso contrario el código de todas las xApps del usuario.

4.3.4 Envio de logs

Servicio disponible en la URL `api/log.php` que permite guardar en la base de datos logs que se produzcan en el cliente.

Recibe los siguientes parámetros:

- **xappid**: Identificador de la xApp que ha producido el log.
- **udid**: UDID del dispositivo donde se ha producido el log.
- **log**: Mensaje que se quiere enviar al servidor.

El sistema está desarrollado de manera que, haciendo uso de Macros de C, cada vez que se realice un log con el método NSLog de Objective-C, se estará mandando toda esta información al servidor para que el desarrollador la pueda consultar posteriormente.

Capítulo 5

Gestión

En este capítulo se detalla la Gestión del Proyecto, mostrando la planificación inicial y final establecida a lo largo del proyecto y un prespuesto de los costes estimados del desarrollo del mismo. Se concluirá con algunas ideas o planes para la monetización de la plataforma.

5.1 Planificación del proyecto

Se ha realizado una planificación detallada del tiempo necesario para la realización del producto final. Inicialmente se mostrará una planificación por fases del proyecto, con una estimación de tiempo necesario para la realización de cada una de ellas. En el punto final se muestra la planificación final resultante.

5.1.1 Planificación inicial

- **Fase Inicial**: Esta primera fase, tenía como finalidad comenzar a sentar las bases del proyecto, desarrollar una idea genérica sobre el proyecto y su envergadura, además de establecer una planificación, para intentar cumplirla. La estimación en horas de esta primera fase fue de 60.
- **Estudio General del sistema**: En esta fase, se pretendía extraer los posibles casos de uso y requisitos de usuario. Estimación de 30 horas.
- **Diseño**: En esta fase se pretendía realizar el diseño y arquitectura de la plataforma, así como definir el lenguaje intermedio y el modo de comunicación entre cliente y servidor. Estimación 120 horas.
- **Implementación**: Esta fase abarca toda la implementación del sistema.
  - **Libreía base:**
    - Parser XML y definición del modelo de datos de la librería. 40 horas.
    - Core Javascript, 30 horas.
    - Sistema de creación de tablas dinámicas con SQLite dependiendo del modelo de datos de la xApp, 40 horas.
    - Sistema para añadir crear vistas a partir de la estructura de datos creada por el parser, 120 horas.
  - **Componentes básicos:**
    - Botones, 20 horas.
    - Label, 10 horas.
    - Campos de texto, 20 horas.
    - Imágenes, 7 horas.
    - Mapas, 26 horas.
    - WebViews, 12 horas.
    - Tablas, 26 horas.
  - **Aplicación web:**
    - Sistema de sesiones con las credenciales del usuario, 6 horas.
    - Clases de acceso a la base de datos, 12 horas.
    - Preparación del editor de texto, 12 horas.

    - Maquetación, 10 horas.
    - Sistema de notificaciones push, 5 horas.

- **Testeo del sistema**: En esta fase se probaría el funcionamiento del sistema, comprobando su correcto funcionamiento. Se estiman 20 horas.
- **Manual de Usuario**: En esta fase se incidía en la creación del manual de usuario, para el proyecto desarrollado. Se estiman 10 horas.
- **Documentación:** En esta fase se pretendía terminar de redactar el documento actual. Estimación 120 horas.
- **Preparación de la presentación**: Tiempo de dicado a preparar la demo y presentación de este proyecto. Se estman 12 horas.

En la figura 5.1 se puede ver el diagrama de Gant del desarrollo del proyecto.

Aunque esta fue la planificación inicial ha sido imposible seguirla, durante la realización del proyecto he compaginado trabajo a jornada completa con las últimas asignaturas de la carreara, por lo que era terriblemente difícil encontra huecos para avanzar el Proyecto. La desviación ha sido de casi dos meses.

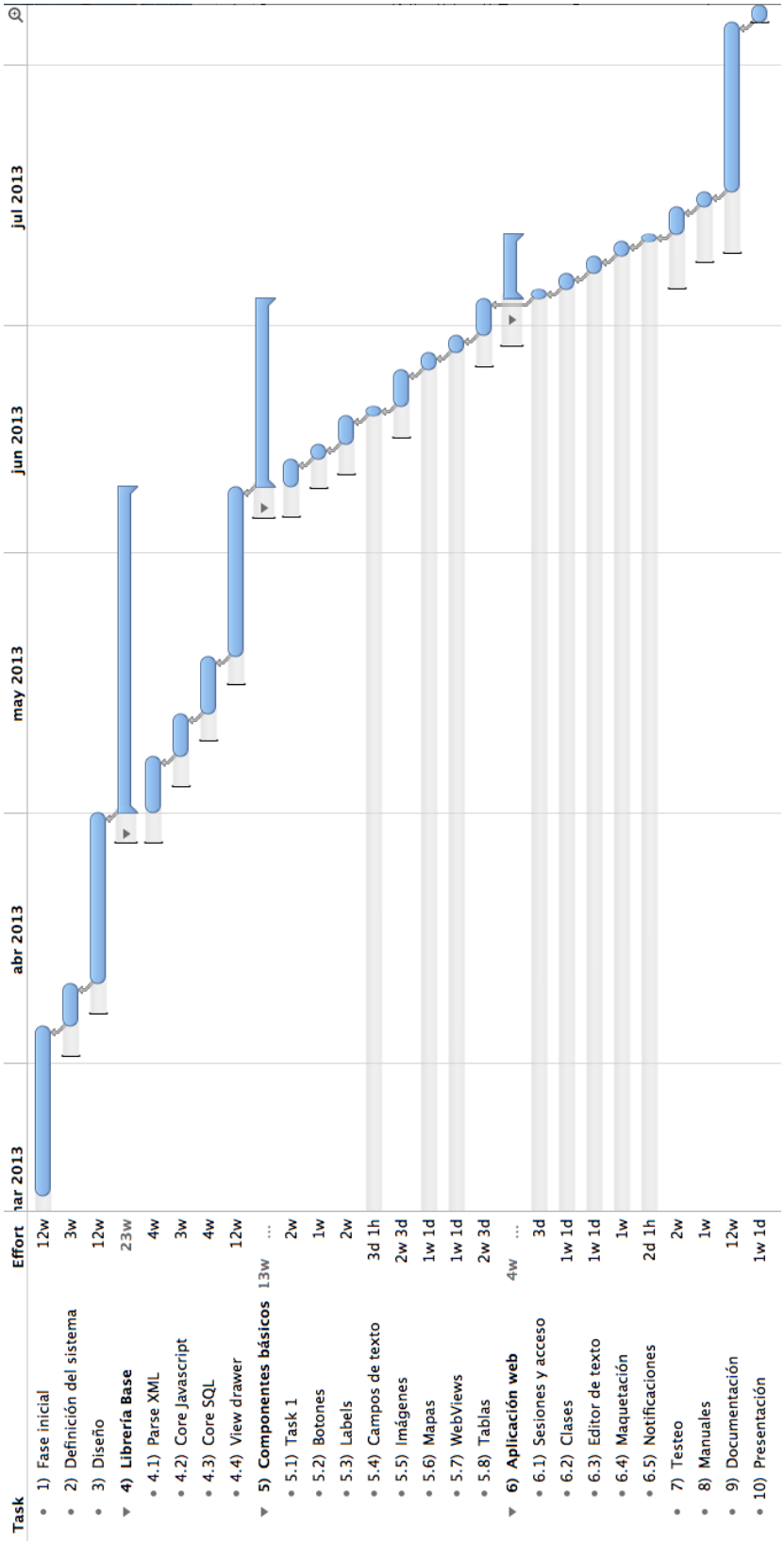

Figura 5.1: **Diagrama de Gant -**

5.2 Presupuesto

En este punto se analizará el coste del proyecto orientativo que se pensó antes de iniciarse el proyecto.

5.2.1 Costes de personal

Los costes de personal o recursos humanos (RRHH) se componen únicamente de los honorarios del Ingeniero Informático encargado del desarrollo del proyecto. Los honorarios correspondientes a un Ingeniero Informático se pueden ver en las bases de cotización de contingencias comunes, fijadas por el Ministerio de Trabajo e Inmigración español.

Concepto	**Horas**	**Honorarios**	**Costes RRHH**
Ingeniero Infrormático	770 horas	45 €/hora	34.650 €

Tabla 5.1: Presuspuesto de RRHH

5.2.2 Costes de hardware

A continuación se especificará un listado del hardware utilizado durante la realización del proyecto junto con su precio para poder estimar el coste que supone el hardware en el mismo.

Notar que sólo se procederá a facturar el hardware que ha sido estrictamente necesario para la realización del proyecto.

Concepto	**Unidades**	**Precio unitario**	**Periodo de amortización**	**Coste**
MacBook Air	1	1800 €	36 meses	350 €
iPhone 4S	1	600 €	24 meses	175 €
Pantalla 22”	1	300 €	48 meses	43,75 €
TOTAL				568,75 €

Tabla 5.2: Presuspuesto de hardware

5.2.3 Costes de sofware y derivados

En este apartado se muestran los costes asociados a los equipos informáticos necesarios para el desarrollo del proyecto y de los recursos software que se han necesitado. El calculo se puede observar en la tabla 5.3.

Concepto	Precio unitario	Periodo amortización	Coste
Omniplan	121 €	12 meses	70 €
Omnigraffle	80 €	12 meses	46,7 €
Creative Suite CS5	999 €	24 meses	287 €
Licencia de desarrollo	80 €	12 meses	46,62 €
Hosting web	12 €	12 meses	7 €
TOTAL			457,32 €

Tabla 5.3: Presuspuesto de sofware y derivados

5.2.4 Presupuesto final del Proyecto

En la tabla 5.4 el cálculo total de los costes del proyecto.

Descripción	Coste Total
Personal	34.650,00 €
Hardware	568,75 €
Software	457,32 €
Total	35.676,07 €

Tabla 5.4: Presuspuesto final

5.3 Modelo de monetización

En esta sección se popondrán alguans ideas de como monetizar este proyecto, al ser un producto sin terminar no se podrá seleccionar ninguna, ya que se tienen datos de uso.

Vender el framework. Consistiría en vender la plataforma a un cliente final. El cliente sería una empresa o similar interesada en desarrollar sus aplicaciones propias para múltiples dispositivos. Aplicaciones para uso interno de la empresa y no para uso particular. Con xAPP esta empresa podría instalar las librerias nativas en cada uno de sus dispositivos de empresa y sincronizarlas con las aplicaciones de un determinado usuario. De esta forma conseguiría que todos sus empleados tengan la aplicación coorporativa en sus dispositivos y aporvecharía todas las características de xAPP. Al ser una venta ad-hoc se podría paquetizar de manera que mostrase los estilos y logos de la empresa compradora.

Cobrar por registro. Cobrando al desarrollador que se registra en xAPP, después este podría publicar tantas aplicaciones como quisiese, y distribuirlas cobrando o gratuitamente a sus clientes. Sería una versión de la venta a empresas más enfocada al pequeño comercio donde no se tiene control del dispositivo final.

Capítulo 6

Conclusión y líneas futuras

En éste capítulo se detallan las conclusiones del Proyecto y las distintas líneas de futuro con las que se puede continuar el desarrollo.

6.1 Conclusión

Empecé este Proyecto hace ya casi un año, y me ha llevado bastante más tiempo del que esperaba, mucho más... Y es que, se hace duro tener que compaginar trabajo y Proyecto, y más cuando lo compaginas con dos trabajos.

Durante el transcurso del año comencé a desarrollar aplicaciónes para móviles con unos compañeros, y después del éxito de algunas de las aplicaciónes acabamos fundando nuestra propia empresa, Sweetbits S.L. De esto es algo de lo que que me siento muy orgulloso, pero ha hecho que en numerosas ocasiones me plantease abandonar este Proyecto de Fin de Carrera para poder centrarme en nuestros proyectos. No sé si habré acertado o no eligiendo terminar la carrera, espero que sí. Y espero poder reengancharme al equipo de Sweetbits ahora, y ver si llegamos a Navidad con una gran apuesta en la que ya llevamos trabajando más de un año.

He aprendido mucho durante el desarrollo de éste Proyecto, sobre todo en lo relativo a como funciona Objective-C a bajo nivel, cosas que no se aprenden con libros de iniciación a la programación para iPhone, por lo que estoy muy satisfecho.

6.2 Objetivos cumplidos

Creo que he conseguido implentar el grueso de la idea inicial, tengo un Framework completamente funcional con el que se pueden desarrollar aplicaciones con muy pocas líneas de código.

La idea de poder desarrollar desde la nube me parece novedosa, se le puede dar una vuelta de tuerca para conseguir aplicaciónes autocontenidas, e incluso poder embeberlas en un IPA; realizar ésto en esta fase alargaría mucho el Proyecto.

Creo que haber realizado éste Proyecto me va a ayudar en el futuro a saber dimensionar desarrollos largos; saber cómo modular la funcionalidad para poder repartirla en el tiempo e incluso poder dividirla para repartirla entre diferentes equipos de trabajo.

6.3 Lineas futuras

Queda mucho trabajo por delante, una de las características que hacen a xAPP un framework diferente a la competencia es la posibilidad de programar módulos complejos que luego puedan ser usados de manera muy sencilla; por este camino se podrían añadir módulos para leer códigos de barras, hacer fotos...

Otro punto a tratar es cómo crear aplicaciones autocontenidas, es decir un único ejecutable con una única aplicación desarrollada con xAPP.

También he pensado en el modo en el que distribuir xApps sin tener que pasar por los Markets, y es construyendo un catálogo propio de aplicaciones, un espacio donde poder listar todas las aplicaciones públicas desarrolladas con xApp de manera que otro usuarios las puedan descargar e instalarlas directamente.

Otro modo de distribuir las plicaciones sería mediante URL o mediante códigos QR. Por ejemplo, un dentista crea una xApp muy sencilla para pedir consulta, entonces cuelga un QR en la puerta de su consulta que al ser leido con la app de xAPP permite a otros usuarios usar esta aplicación.

Por delante queda portar la librería creada para iOS al resto de plataformas, de manera que xAPP no solo sea un concepto de framework de desarrollo multiplataforma, sino algo real.

Bibliografía

[1] JSON Javascript Object Notation: http://www.json.org/

[2] SQLite http://www.sqlite.org/

[3] Mugunth Kumar: iOS 5 Programming Pushing the Limits. Editorial Wiley; 2 edition (December 20, 2011)

[4] Stephen G. Kochan: Programming in Objective-C. 4th Edition

[5] XML http://www.w3schools.com/xml/

[6] Robert Mcnally: The Code Commandments: Best Practices for Objective-C Coding. http://ironwolf.dangerousgames.com/blog/archives/913

[7] Extensible Markup Language (XML). http://www.w3.org/XML/

[8] Roy Fielding: Representational State Transfer (REST). http://www.service-architecture.com/web-services/articles/representational_state_transfer_rest.html

[9] Documentación de Apple sobre iPhone. https://developer.apple.com/iphone/

[10] PhoneGap framework. http://phonegap.com/

[11] Senecha Touch. http://www.sencha.com/

[12] jQuery Mobile. http://jquerymobile.com/

[13] Javascript. https://developer.mozilla.org/en/About_JavaScript

[14] Appcelerator Studio. http://www.appcelerator.com/products/titanium-studio/

[15] Appcelerator Titanium Mobile http://developer.appcelerator.com/apidoc/mobile/

[16] iOS Sandbox. http://developer.apple.com/library/ios/DOCUMENTATION/iPhone/Conceptu
oneOSProgrammingGuide/TheiOSEnvironment/TheiOSEnvironment.html

Anexo A

Manual de usuario iOS

Este manual es una simple guia de uso de la aplicación para iOS. Su uso no tiene complicación posible pero aquí queda una pequeña reseña de su uso.

A.1 Instalación de la app

En este momento la única forma de instalar la app es descargando el IPA del portal de xAPP, este es un IPA para distibción Enterprise por lo que no es necesario que esté publicado en el AppleStore, aunque se estudia la forma de abrir esa linea de negocio.

A.2 Instrucciones de uso

Una vez instalada la aplicación aparecerá el icono de la misma en el springboard de nuestro dispositivo. Al abrirla s enos pedirá que hagamos login con nuestra cuenta de xAPP.

Figura A.1: **Pantalla de login** -

Trás realizar el login correctamente aparecera la pantalla de aplicaciones con las aplicaciones que hemos creado en el portal web. Podemos abrir cualquiera de ellas haciendo tap sobre su título o actualizar las aplicaciones pulsando en el botón de la esquina superior derecha.

Figura A.2: **Listado de xApps** -

Una vez entremos en una xApp para volver al menú inicial debemos agitar el teléfono que nos devolverá a la pantalla de selección de aplicaciones.

Cuando se produzca alguna modificación de una de las xApps que tenemos instaladas en el portal web, nos llegará una notificación indicando esto.

Figura A.3: **Notificación push** -

Anexo B

Manual de usuario de la Web

Esta es una guia que detalla las opciones y el modo de uso de la página web de xAPP, desde donde el desarrollador puede gestionar y editar sus aplicaciones.

B.1 Registro y login

La figura B.1 es la primera pantalla que se muestra tras acceder a la página web. Desde el usuairo puede hacer login o registrarse si no tiene una cuenta ya creada.

Los únicos datos necesarios para registrarse sun un email y una contraseña.

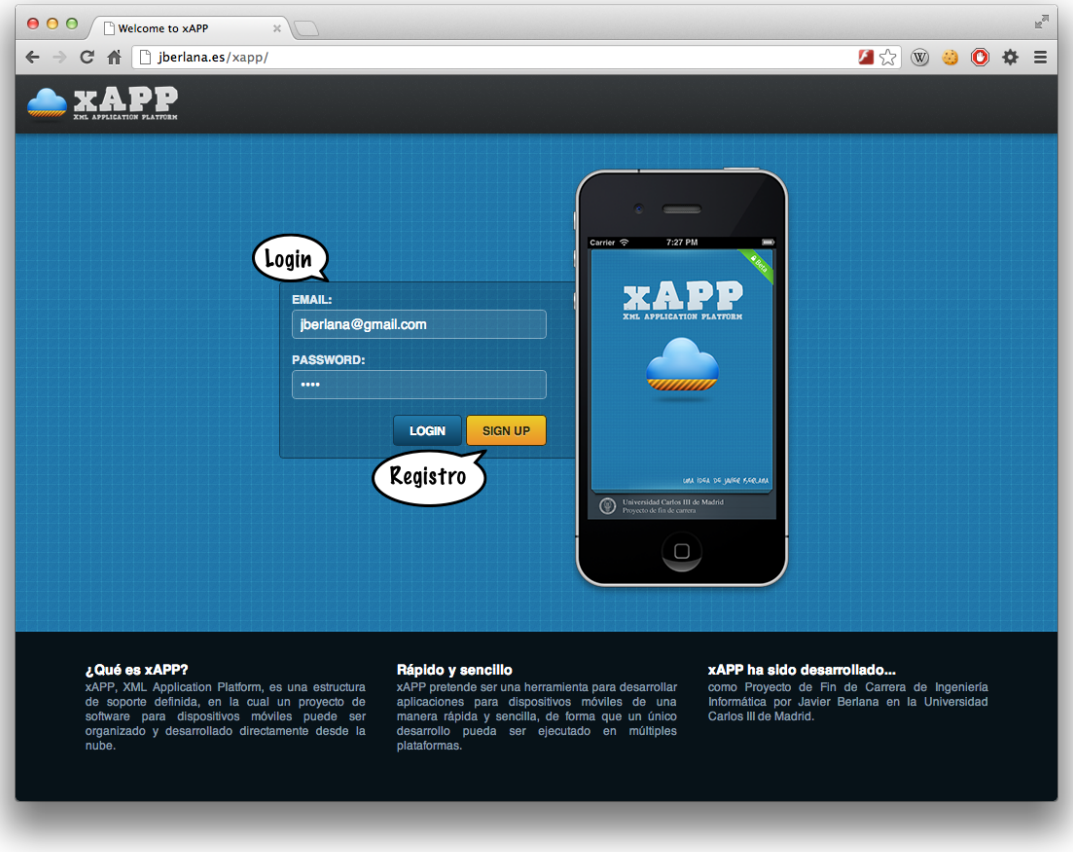

Figura B.1: **Página de login** -

B.2 Listado de xApps y logs

Esta es la pantalla principal del portal web. Se muestra tras logearse correctamente desde la pantalla anterior y desde ella el usuario puede crear, editar y eliminar sus xApps, además de ver los logs que éstas están prodiciendo en los dispositivos dónde ya están instaladas.

En la figura B.2 se puede ver la funcionalidad de cada uno de los elementos de la web.

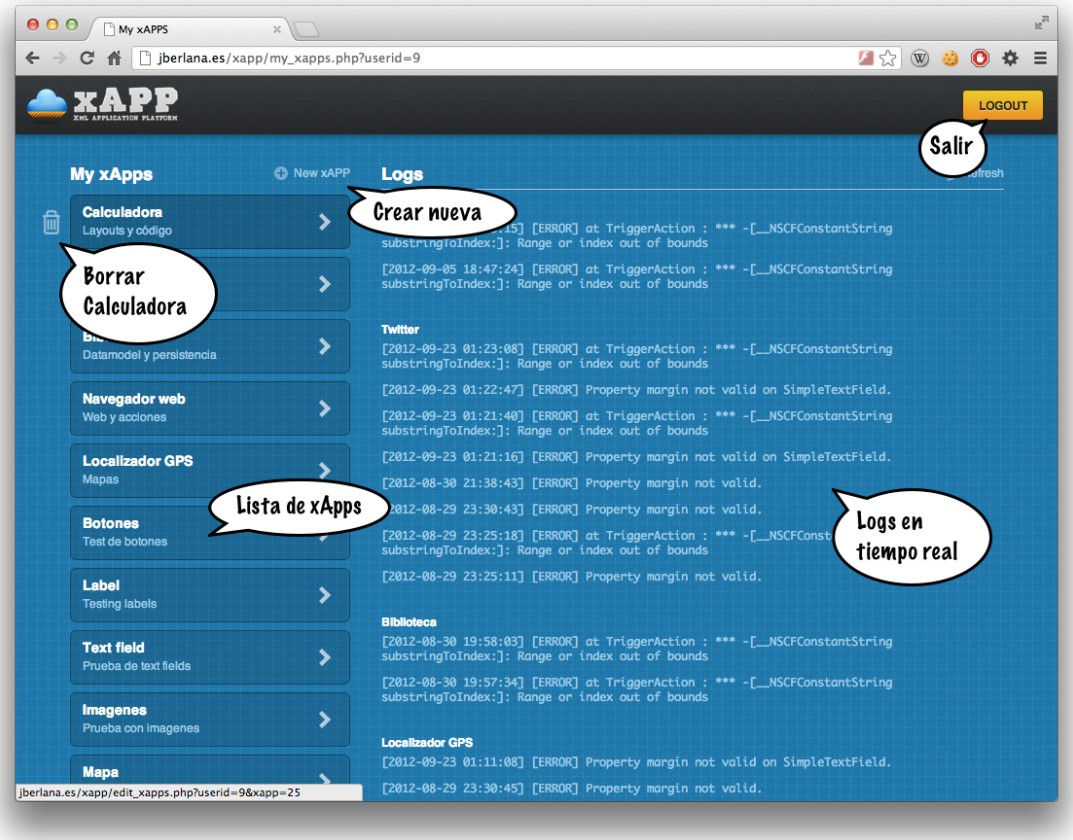

Figura B.2: **Página principal del portal web -**

B.3 Creación y edición de xApps

La figura B.3 muestra la pantalla de edición de código. Esta pantalla muestra tres pestañas 'XAPP', 'DATA' y 'CODE' cada una de ellas sirve para editar las vistas de la xAPP, el modelo de datos o el código. Además se muestran otros cuatro botones, para guardar el proyecto, cerrarlo y deshacer y rehacer acciones.

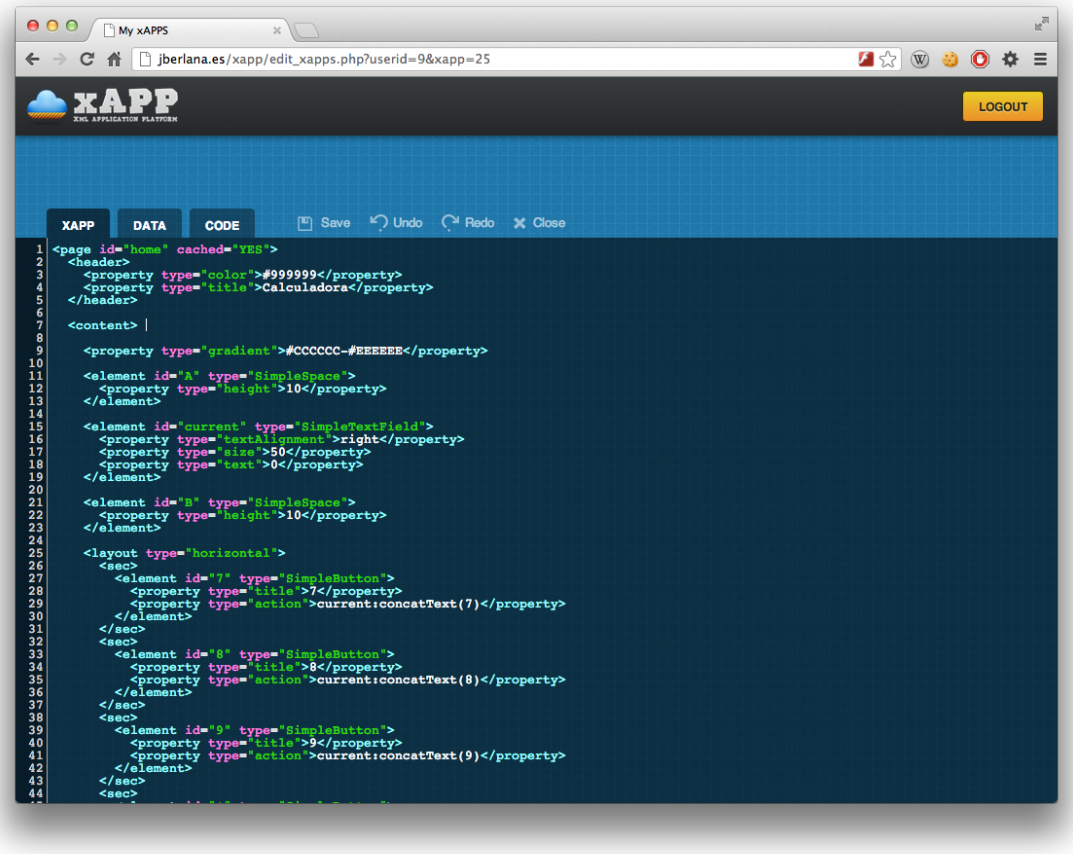

Figura B.3: **Editor de código** -

Anexo C

Guía de desarrollo de xAPPs

Este manual pretende servir de guía para el desarrollador que haga uso de xAPP par acrear sus aplicacionesd móviles.

C.1 Estructura de una app en xAPP

La estructura de un xApp viene definida por un esquema XML, ver fragmento de código C.1, que define el esqueleto básico de una xApp, nos encontramos con un elemento principal `app` que puede contener en su interior de 1 a n `page`, la página raiz de la xApp siempre debe ser nombrada **home** y cada una de estas páginas definirán cada una de las vistas de la aplicación, además la app podrá tener un elemento `datamodel` y `code` ambos opcionales, los cuales definirán el modelo de datos y el código interno de la xApp respectivamente.

Listing C.1: Esqueleto de una xApp

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<app>
  <page id='home' cached='YES'>
    ...
  </page>
  <page id='other' cached='NO'>
    ...
  </page>
  ...
  <datamodel>
    ...
  </datamodel>
  <code>
    ...
  </code>
</app>
```

C.1.1 Las vistas

Las vistas de la xApp vienen definidas por elementos `page` contiene tres elementos principales que a su vez pueden contener sub-elementos:

- `objects` Declara objetos que se utilizarán dentro de la página.
- `header` que define el estilo y contenido de la barra de navegación. El header puede incluir una serie de propiedades para definir colores y fuentes, así como botónes que se situarán en la parte derecha del navigatión bar. La figura 3.10 muestra la arquitectura de una página en xAPP.
- `content` que define el contenido de la página.

  El `content` de una xAPP está formado por `layout`, `sec` y `element` estos son layouts, secciones y elementos. Un layout define la forma en la que se alinean las

vistas en su interior, por defecto XAPP alineará todos los elementos en sucesión uno debajo del otro, haciendo uso de `<layout type='horizontal'>` conseguiremos que esta alineación se realice horizontalmente. Una layout además puede subdividirse en secciones que a su vez pueden contener otros layouts o elementos.

- `footer` elemento opcional, al igual que el `header` que define contenido fijo en la parte baja de la pantalla.

En el fragmento de código C.2 se puede ver cómo se estructuran estos elementos dentro de la página. Tanto `content` cómo `header` pueden incluir propiedades que definen el aspecto de los mismos, cómo puede ser un color de fondo o degradados.

Listing C.2: Esqueleto de una página

```
<page id="home" cached="NO">
  <objects>
    book : null;
  </objects>

  <header>
    <property type="title">Library</property>
    <element id="barBut1" type="Add">
      <property type="alignment">right</property>
      ...
    </element>
  </header>

  <content>
    <property type="color">#cccccc</property>
    <element id="tableBooks" type="SimpleTable">
      <property type="height">100%</property>
      ...
    </element>
    ...
  </content>

  <footer>
  ...
  </footer>
</page>
```

El `content` incluye, además, la sucesión de elementos de los que se componga la página, estos elementos dependen de la versión que se esté utilizando del framework. Cada uno de estos elementos define vistas de diferentes tipos, controladores, botones,

labels de texto...

La arquitectura general de un elemento se muestra en el fragmento de código C.3. Todo elemento tiene dos atributos obligatorios:

- `id`: Identificado único del elemento, podemos verlo cómo el nombre que se le da al objeto del tipo Element, y será el nombre con el que se le referencie desde otros objetos.
- `type`: Determina el tipo de elemento que estamos creando, viene dado por el nombre del módulo del elemento en xAPP, por ejemplo, SimpleButton, SimpleLabel o SimpleTextField. Dependiendo del tipo tendrá unas propiedades disponibles u otras.

Listing C.3: Definición de un elemento

```
<element id='0' type='FooObject'>
  <property type='text'>{object:book.title}</property>
  <property type='title'>0</property>
  <property type='action'>
    current:concatText(0);
    temp:editText({var:home.current.text});
  </property>
</element>
```

Todo elemento tendrá de 1 a n `property`, y el contenido de una `property` puede ser de tres tipos diferentes.

- Descriptivas básicas: Definen propiedades del elemento con constantes, en el ejemplo `<property type='title'>0</property>`
- Descriptivas complejas: Definen propiedades del elemento con datos variables o dependientes de otro elemento de la xApp.

  Las tenemos de dos tipos, las que cogen el valor de un objeto de la xApp: {`object:book.title`}. Donde `object` indica que el valor se encuentra en el modelo de datos y `book.title` que se trata de la columna de `title` de la tabla `book`.

  Las que toman el valor de otro elemento de la xAPP por ejemplo:

  {`var:home.current.text`} Dónde `var` indica que el valor será tomado de otro elemento y `home.current.text` la sucesión: id de página, id de elemento y propiedad del elemento.

- El último tipo corresponde a las propiedades de tipo action, estas propiedades se pueden dar en botones o elementos que quieran lanzar una determinada acción cuando se produzca un determinado evento.

  La estructura de las acciones es la siguiente:

  `[id de la página]<id del elemento>:<nombre del método>([parámetros]);`

  Los parámetros vienen definidos por la estructura vista en las propiedades descriptivas complejas del punto anterior.

C.1.2 El modelo de datos

Aquellas xApps que requieran de persistencia de datos o necesiten prepoblar una seríe de páginas iguales con diferentes datos sin duplicar la vista, es decir hacer uso del patrón Modelo-Vista-Controlador, utilizando una estructura maestro-detalle pueden hacer uso de esta característica.

Un ejemplo de data model es el siguiente que se muestra en el fragmento de código C.4

Listing C.4: Ejemplo de datamodel

```
<datamodel>
  <book>
    <id>1</id>
    <title>El nombre del viento</title>
    ...
  </book>
  <book>
    <id>2</id>
    <title>El temor de un hombre sabio</title>
    ...
  </book>
  ...
</datamodel>
```

El datamodel sirve tanto para definir el esquema del modelo de datos, como para poblarlo, si únicamente se quiere definir el esquema basta con crear las tablas con los elementos vacíos, si queremos poblarlos crearemos tantos elementos como filas queramos insertar en nuestra tabla. En el ejemplo del fragmento de código C.4 se ha creado la tabla `book` que tiene las columnas `id` y `title` y se han insertado dos filas de libros diferentes.

C.1.3 El código

El código se incluye en la sección `code` de la xApp, el código permitido son funciones Javascript, cómo la del siguiente ejemplo:

Listing C.5: Ejemplo de código

```
<code>

function getResult(temp, opp, current){
    var result = eval(temp + opp + current);
    to_xapp('home.current:setText('+result+')');
}

</code>
```

En el ejemplo se define una función en javascript que recibe tres valores por parámetro y devuelve un resultado por medio de la función interna `to_xapp()`. Para llamar a esta función desde un elemento `action` de la xApp se utiliza la nomenclatura:

```
app:<nombre del método>([parámetros]);
```

Por ejemplo, en el caso del código definido en C.5 la llamada se realizará de la siguiente forma:

```
app:getResult(4,'+',3);
```

En el apartado para código de la xAPP podemos definir variables locales, funciones recursivas, en definitiva, cualquier código javascript válido. Como hemos visto en la sección 4.1.4 JS Core, xAPP corren una instancia del interprete de Javascript del navegador nativo de la plataforma que se está utilizando, y ejecuta en él el código introducido. Siendo el punto de entrada la llamada por medio de `app:<método>();` y la salida la ejecutada en la función `to_xapp()`.

C.2 Componentes

Ahora se detallaran los componentes que admite xAPP en la versión actual, y las propiedades y opciones qu eadmiten cada uno de estos.

C.2.1 Botón

Elementos con el que el usuario puede interactuar para desencadenar una determinada acción.

xAPP permite crear botones con un estilo personalizado en muy pocas lieneas, dando

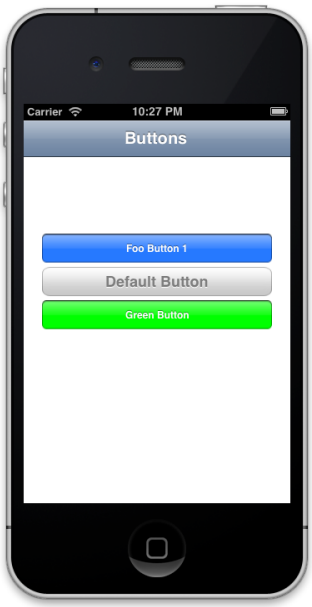

Figura C.1: **Ejemplo de botón** -

resultados rápidos y mucho más visuales que los botones nativos. Pudiendo aplicar colores por código RGB para personalizar su color, opción que no está disponible en el SDK nativo.

Las propiedades que admite un botón son las siguientes:

- **title**: Define el texto que aparecerá en el botón.
- **color**: Define el color del botón, debe ser un código RGB.
- **margin**: Pixeles en blanco que dejará el botón a la derecha e izquierda.
- **height**: Altura del botón.
- **action**: Acción que se realizará cuando el botón reciba un evento de tipo tap_inside.

Listing C.6: Código para un botón

```
1 <element id="button1" type="SimpleButton">
```

```
2     <property type="margin">20</property>
3     <property type="height">40</property>
4     <property type="title">Foo Button</property>
5     <property type="color">#4099FF</property>
6     <property type="action">
7       home:foo(2);
8     </property>
9   </element>
```

El botón incluido en xAPP además permite realizar tareas cómo mandar un email o twitear simplemente con definir la acción del tipo:

- `tweet()`: Para lanzar un Tweet.
- `mail()`: Para mandar un email.

Ambas reciben por parámetro el texto que se quiere enviar.

C.2.2 Label

Un label es un texto corto, en xAPP de menos de una linea, comúnmente usado para describir otro elemento o mostrar información en pantalla.

Las propiedades que admite un botón son las siguientes:

- **text**: Define el texto que aparecerá en el label.
- **color**: Define el color del label, debe ser un código RGB.
- **alignment**: Alineación del label en la página, (center, left o right).
- **font**: Tipo de fuente que se utilizará.
- **size**: Tamaño de la fuente.
- **alpha**: De 0 a 1, define la opacidad del label

Este control adicionalmente puede recibir cómo property cualquier atributo válido del elemento `UILabel` de iOS. Esto es así por el modo en el que ha sido implementado el set de los properties no normalizados. Observese el fragmento de código 4.7.

Listing C.7: Código para label

```
<element id="myLabel" type="SimpleLabel">
  <property type="text">Hello World!</property>
  <property type="color">#333399</property>
  <property type="alignment">center</property>
  <property type="font">Helvetica-Bold</property>
  <property type="size">50</property>
</element>
```

Figura C.2: **Ejemplo de label** -

C.2.2.1 Campo de texto

Es el elemento que utilizará el usuario para introducir información en la aplicación en forma de texto.

- **text**: Define el texto que aparecerá en el TextField.
- **textalignment**: Alineación del texto en el TextField, (center, left o right).
- **font**: Tipo de fuente que se utilizará.
- **size**: Tamaño de la fuente.
- **mapping**: Permite mapear el textfield con un objeto de la página.

Figura C.3: **Ejemplo de textfield** -

Es importante destacar la propiedad `mapping`, que nos permite mapear un TextField a un objeto definido en la página, de manera que si este objeto pertenece al modelo de datos y está marcado como persistente, nos permitirá modificar su valor el la DB una vez el usuario termine de escribir.

Listing C.8: Código para SimpleTextField

```
<element id="text2" type="SimpleTextField">
  <property type="placeholder">Foo field 2</property>
  <property type="textalignment">center</property>
  <property type="font">Helvetica-Bold</property>
  <property type="size">30</property>
  <property type="mapping">{object:book.author}</property>
</element>
```

En el fragmento de código anterior se mapea el valor del SimpleTextField con el atributo `author` del objeto `book` definido en esa misma página.

Es importante destacar que SimpleTextField es capaza de recibir acciones, esto signifivca que puede ser invocado por otros elemento con el fin de modificar su estado. Estas acciones son:

- **concatText**: Concatena texto al final del texto actual.
- **setText**: Cambia el texto completo en el textfield.

Por ejemplo:

Listing C.9: Código para SimpleTextField

```
<element id="A" type="SimpleTextField">
  <property type="textAlignment">right</property>
  <property type="size">50</property>
  <property type="text">0</property>
</element>

<element id="B" type="SimpleButton">
  <property type="title">B</property>
  <property type="action">current:concatText(0)</property>
</element>
```

En este fragmento de código tenemos un botón B que al ser pulsado concatena un 0 al final del texto en A.

C.2.3 Imágenes

Las imágenes son un elemento básico de cualquier aplicación móvil que no podían faltar en xAPP, actualmente el entorno web no permite la subida de ficheros por lo que las imágenes se cargan desde URLS de internet.

Las propiedades son:

- **image**: Para indicar la URL de la imágen a cargar.
- **width**: Ancho de la imagen.
- **height**: Alto de la imagen.
- **align**: Alineación de la imagen en la página, (center, left o right).

Figura C.4: **Ejemplo de imagen en una xApp** -

C.2.4 Mapas

Elemento usado para geoposicionar al usuario o mostrar un punto de interés, internamente hace uso de los mapas de Google, através del componente `MKMapView`. Las propiedades que admite son las siguientes:

- **height**: Altura que ocupará el mapa en la página actual.
- **border**: YES o NO si queremos que el mapa se muestre con borde.

Pero además puede recibir dos tipos de acciones:

- **goTo([destino],[origen])**: Mostrara la ruta entre los dos puntos.
- **lookFor([dirección])**: Realizará una búsqueda de la dirección escrita y mostrará en el mapa la primera posición devuelta por GOOGLE.

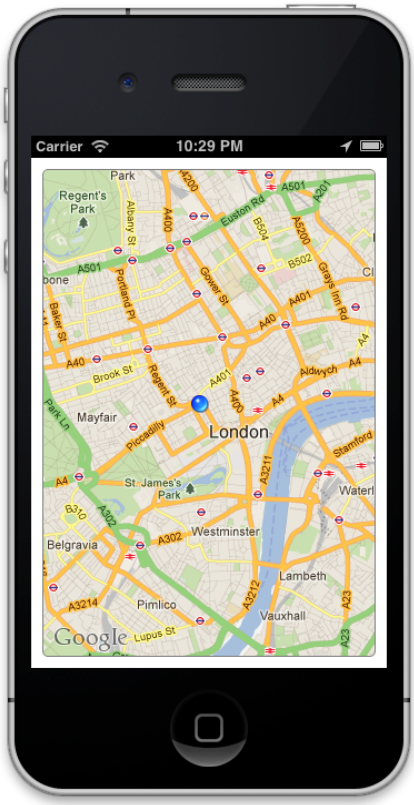

Figura C.5: **Ejemplo de mapView** -

xAPP simplifica notablemente el uso de mapas en una aplicación móvil, en pocas lineas conseguimos lo que en desarrollo nativo llevaría unos miles de lineas y el uso de

numerosas clases y librerías. iOS por defecto no ofrece la posibilidad de hacer búsquedas ni pintar rutas sobre los mapas por lo que esta funcionalidad ha sido desarrollada desde cero.

C.2.5 Webview

SimpleWebView es el módulo incluido en xAPP que se encarga de colocar una vista web directamente dentro de tus xApps.

Figura C.6: **Ejemplo de textfield** -

Los parámetros que acepta este componente son los siguientes:

- **url**: La URL de la web a cargar.
- **height**: Altura que ocupará el mapa en la página actual.
- **border**: YES o NO si queremos que el mapa se muestre con borde.

Y además puede recibir los siguientes tipos de acciones:

- **goTo([url])**: Cargará la págian indicada en el webView.
- **goBack()**: Típica acción atrás de un navegador web.
- **goForward()**: Volverá hacia la pagina cargada antes de pulsar atrás.

C.2.6 Tablas

En toda aplicación móvil es necesario mostrar un listado de elementos, por lo que en xAPP no podía faltar esta funcionalidad.

SimpleTableView necesita de un modelo de datos definido en la xApp para poder funcionar ya que lo utilizará cómo `datasource`. El siguiente fragmento de código muestra cómo se define una tabla de datos:

Listing C.10: Código para tablas

```
<element id="tableBooks" type="SimpleTable">
  <property type="height">100%</property>
  <property type="width">100%</property>
  <property type="table">book</property>
  <property type="title">title</property>
  <property type="subtitle">author</property>
  <property type="action">home:pushPage(detail)</property>
</element>
```

Los parámetros que puede recibir son los siguientes:

- **height**: Porcentaje de altura que ocupará en la tabla en la pantalla.
- **width**: Porcentaje de anchura que ocupará en la tabla en la pantalla.
- **table**: Nombre del tabla del modelo de datos que se utilizará para rellenar la tabla, en el ejemplo podemos ver como se utilizará la tabla llamada `book`.
- **title**: Nombre del atributo de la tabla seleccionada que se utilizará cómo titulo principal para celda, en el caso del ejemplo corresponde a `title`.
- **subtitle**: Nombre del atributo de la tabla seleccionada que se utilizará cómo subtitulo para celda, en el caso del ejemplo corresponde a `author`.
- **action**: Acción que se lanzará cuando el usuario haga tap sobre una determinada celda. Además la tabla pasara cómo parámetro el objeto sobre el que se ha hecho tab, que la siguiente página recibirá con:

```
<objects>
  book : {prev.book};
```

```
3   </objects>
```

- **delete**: True en caso de que queramos que el usuario pueda eliminar celdas de la tabla. Al eliminarse una celda se eliminará también la fila relacionada el el modelo de datos.

Figura C.7: **Ejemplo de tabla** -

Anexo D

xApps de ejemplo

D.1 Calculadora

Esta xApp muestra como haciendo uso de layouts, botones, código javascript y un textField podemos construir una calculadora.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>

<app>
  <page id="home" cached="YES">
    <header>
           <property type="color">#999999</property>
      <property type="title">Calculadora</property>
    </header>

    <content>

           <property type="gradient">#CCCCCC-#EEEEEE</property>

           <element id="A" type="SimpleSpace">
               <property type="height">10</property>
           </element>

           <element id="current" type="SimpleTextField">
               <property type="textAlignment">right</property>
               <property type="size">50</property>
               <property type="text">0</property>
           </element>

```

```
24              <element id="B" type="SimpleSpace">
25                  <property type="height">10</property>
26              </element>
27
28              <layout type="horizontal">
29                  <sec>
30                      <element id="7" type="SimpleButton">
31                          <property type="title">7</property>
32                          <property type="action">current:concatText(7)</property>
33                      </element>
34                  </sec>
35                  <sec>
36                      <element id="8" type="SimpleButton">
37                          <property type="title">8</property>
38                          <property type="action">current:concatText(8)</property>
39                      </element>
40                  </sec>
41                  <sec>
42                      <element id="9" type="SimpleButton">
43                          <property type="title">9</property>
44                          <property type="action">current:concatText(9)</property>
45                      </element>
46                  </sec>
47                  <sec>
48                      <element id="+" type="SimpleButton">
49                          <property type="title">+</property>
50                          <property type="action">
51                              temp:editText({var:home.current.text});
52                              opp:editText(+);
53                              current:setText();
54                          </property>
55                      </element>
56                  </sec>
57              </layout>
58
59              <layout type="horizontal">
60                  <sec>
61                      <element id="4" type="SimpleButton">
62                          <property type="title">4</property>
63                          <property type="action">current:concatText(4)</property>
64                      </element>
```

```
65                  </sec>
66                  <sec>
67                      <element id="5" type="SimpleButton">
68                          <property type="title">5</property>
69                          <property type="action">current:concatText(5)</property>
70                      </element>
71                  </sec>
72                  <sec>
73                      <element id="6" type="SimpleButton">
74                          <property type="title">6</property>
75                          <property type="action">current:concatText(6)</property>
76                      </element>
77                  </sec>
78                  <sec>
79                      <element id="-" type="SimpleButton">
80                          <property type="title">-</property>
81                          <property type="action">
82                              temp:editText({var:home.current.text});
83                              opp:editText(-);
84                              current:setText();
85                          </property>
86                      </element>
87                  </sec>
88              </layout>
89
90              <layout type="horizontal">
91                  <sec>
92                      <element id="1" type="SimpleButton">
93                          <property type="title">1</property>
94                          <property type="action">current:concatText(1)</property>
95                      </element>
96                  </sec>
97                  <sec>
98                      <element id="2" type="SimpleButton">
99                          <property type="title">2</property>
100                         <property type="action">current:concatText(2)</property>
101                     </element>
102                 </sec>
103                 <sec>
104                     <element id="3" type="SimpleButton">
105                         <property type="title">3</property>
```

```
106                         <property type="action">current:concatText(3)</property>
107                     </element>
108                 </sec>
109                 <sec>
110                     <element id="x" type="SimpleButton">
111                         <property type="title">x</property>
112                         <property type="action">
113                             temp:editText({var:home.current.text});
114                             opp:editText(*);
115                             current:setText();
116                         </property>
117                     </element>
118                 </sec>
119             </layout>
120 
121             <layout type="horizontal">
122                 <sec>
123                     <element id="0" type="SimpleButton">
124                         <property type="title">0</property>
125                         <property type="action">current:concatText(0)</property>
126                     </element>
127                 </sec>
128                 <sec>
129                     <element id="C" type="SimpleButton">
130                         <property type="title">C</property>
131                         <property type="action">
132                             current:setText();
133                             opp:editText();
134                             temp:editText();
135                         </property>
136                     </element>
137                 </sec>
138                 <sec>
139                     <element id="=" type="SimpleButton">
140                         <property type="title">=</property>
141                         <property type="action">
142                             app:getResult({var:home.temp.text},'{var:home.opp.
                                    text}',{var:home.current.text});
143                         </property>
144                     </element>
145                 </sec>
```

```
146                 <sec>
147                     <element id="/" type="SimpleButton">
148                         <property type="title">/</property>
149                         <property type="action">
150                             temp:editText({var:home.current.text});
151                             opp:editText(/);
152                             current:setText();
153                         </property>
154                     </element>
155                 </sec>
156             </layout>
157 
158             <element id="temp" type="SimpleLabel">
159                 <property type="alpha">0</property>
160             </element>
161 
162             <element id="opp" type="SimpleLabel">
163                 <property type="alpha">0</property>
164             </element>
165 
166     </content>
167 
168   </page>
169 
170     <code>
171 
172         function getResult(temp,opp,current){
173             var result = eval(temp + opp + current);
174             to_xapp('home.current:setText('+result+')');
175         }
176 
177     </code>
178 
179 </app>
```

Figura D.1: **xApp calculadora** -

D.2 Biblioteca

Esta xApp muestra como haciendo uso del modelo de datos, las vistas de tablas podemos construir una xApp que almacena datos de interés del usuario completamente editables.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>

<app>
    <datamodel>

        <book>
            <id>1</id>
            <title>El nombre del viento</title>
            <author>Patrick Rothfuss</author>
            <abstract>Parte 1</abstract>
            <year>2007</year>
        </book>

        <book>
            <id>2</id>
            <title>El temor de un hombre sabio</title>
            <author>Patrick Rothfuss</author>
            <abstract>Parte 2</abstract>
            <year>2011</year>
        </book>

    </datamodel>

    <page id="home" cached="NO">

        <header>
            <property type="title">Library</property>

            <element id="barBut1" type="Add">
                <property type="alignment">right</property>
                <property type="action">pushPage:new</property>
            </element>
        </header>

        <content>
            <property type="color">#cccccc</property>
```

```
37
38              <element id="tableBooks" type="SimpleTable">
39                  <property type="height">100%</property>
40                  <property type="width">100%</property>
41                  <property type="table">book</property>
42                  <property type="title">title</property>
43                  <property type="subtitle">author</property>
44                  <property type="action">home:pushPage(detail)</property>
45              </element>
46
47          </content>
48
49          <footer>
50          </footer>
51      </page>
52
53      <page id="detail" cached="NO">
54          <objects>
55              book : null;
56          </objects>
57
58          <header>
59              <property type="title">Book</property>
60
61              <element id="barBut1" type="Compose">
62                  <property type="alignment">right</property>
63                  <property type="action">pushPage:edit</property>
64              </element>
65          </header>
66
67          <content>
68
69              <element id="labelTitle" type="SimpleLabel">
70                  <property type="text">{object:book.title}</property>
71              </element>
72
73              <element id="labelAuthor" type="SimpleLabel">
74                  <property type="text">{object:book.author}</property>
75              </element>
76
77              <element id="labelYear" type="SimpleLabel">
```

```
78                  <property type="text">{object:book.year}</property>
79              </element>
80
81          </content>
82
83          <footer>
84          </footer>
85      </page>
86
87
88        <page id="edit" cached="NO">
89          <objects>
90              book : {prev.book};
91          </objects>
92
93          <header>
94              <property type="title">Editar</property>
95          </header>
96
97          <content>
98
99              <element id="editAuthor" type="SimpleTextField">
100                 <property type="placeholder">Author name</property>
101                 <property type="mapping">{object:book.author}</property>
102             </element>
103
104             <element id="editTitle" type="SimpleTextField">
105                 <property type="placeholder">Book title</property>
106                 <property type="mapping">{object:book.title}</property>
107             </element>
108
109             <element id="editYear" type="SimpleTextField">
110                 <property type="placeholder">Year</property>
111                 <property type="mapping">{object:book.year}</property>
112             </element>
113
114             <element id="-" type="SimpleButton">
115                 <property type="title">Save to DB</property>
116                 <property type="color">blue</property>
117                 <property type="action">
118                     edit:syncDB(book);
```

```
119                     edit:popPage();
120                 </property>
121             </element>
122 
123         </content>
124 
125     </page>
126 
127       <page id="new" cached="NO">
128         <objects>
129             book : book(new);
130         </objects>
131 
132         <header>
133             <property type="title">Crear</property>
134         </header>
135 
136         <content>
137 
138             <element id="newAuthor" type="SimpleTextField">
139                 <property type="placeholder">Author name</property>
140                 <property type="mapping">{object:book.author}</property>
141             </element>
142 
143             <element id="newTitle" type="SimpleTextField">
144                 <property type="placeholder">Book title</property>
145                 <property type="mapping">{object:book.title}</property>
146             </element>
147 
148             <element id="newYear" type="SimpleTextField">
149                 <property type="placeholder">Year</property>
150                 <property type="mapping">{object:book.year}</property>
151             </element>
152 
153             <element id="newAbstract" type="SimpleTextField">
154                 <property type="placeholder">Abstract</property>
155                 <property type="mapping">{object:book.abstract}</property>
156             </element>
157 
158             <element id="-" type="SimpleButton">
159                 <property type="title">Save to DB</property>
```

```
160                 <property type="action">
161                     new:addDB(book);
162                     new:popPage();
163                 </property>
164             </element>
165 
166         </content>
167 
168   </page>
169 
170 </app>
```

Figura D.2: **xApp biblioteca** -

D.3 Buscador GPS

En esta xApp tenemos un GPS con buscador de direcciones y calculador de rutas en unas 50 lineas.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>

<app>
  <page id="home" cached="YES">
    <header>
           <property type="color">#333399</property>
      <property type="title">Buscador</property>
    </header>

    <content>

           <element id="10" type="SimpleSpace">
               <property type="height">10</property>
           </element>

           <element id="place" type="SimpleTextField">
               <property type="placeholder">Lugar</property>
               <property type="margin">10</property>
           </element>

           <layout type="horizontal">
               <sec>
                   <element id="button1" type="SimpleButton">
                       <property type="margin">10</property>
                       <property type="title">Buscar</property>
                       <property type="action">map:lookFor({var:home.place.text
                           })</property>
                   </element>
               </sec>
               <sec>
                   <element id="button2" type="SimpleButton">
                       <property type="margin">10</property>
                       <property type="title">Como llegar</property>
                       <property type="action">map:goTo({var:home.place.text},
                           currentLocation)</property>
                   </element>
```

```
35                </sec>
36            </layout>
37
38            <element id="map" type="SimpleMap">
39                <property type="height">320</property>
40                <property type="border">YES</property>
41                <property type="pins">{JSON OBJECT}</property>
42        </element>
43
44      </content>
45
46          <footer>
47      </footer>
48
49    </page>
50  </app>
```

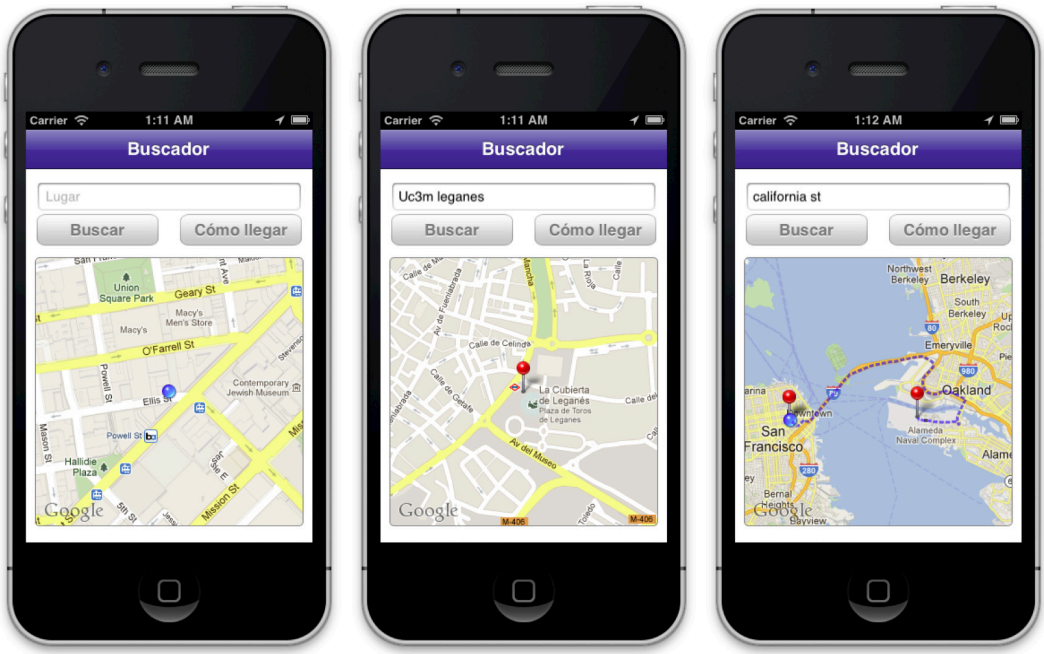

Figura D.3: **xApp Localizador GPS** -

D.4 Web browser

Esta xApp hace uso de botones, layouts, un webView y un textFiled conectado a las acciones de los botones para crear un navegador web completamente funcional en tan solo 44 lineas de código.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>

<app>
  <page id="home" cached="YES">
    <content>

            <layout type="horizontal">
                <sec>
                    <property type="width">35</property>
                    <element id="button1" type="SimpleButton">
                        <property type="title"><-</property>
                        <property type="action">web:goBack()</property>
                    </element>
                </sec>
                <sec>
                    <property type="width">35</property>
                    <element id="button3" type="SimpleButton">
                        <property type="title">-></property>
                        <property type="action">web:goForward()</property>
                    </element>
                </sec>
                <sec>
                    <property type="width">100%</property>
                    <element id="url" type="SimpleTextField">
                        <property type="placeholder">Introduzca la url</property
                            >
                    </element>
                </sec>
                <sec>
                    <property type="width">35</property>
                    <element id="button2" type="SimpleButton">
                        <property type="title">Ir</property>
                        <property type="action">web:goTo({var:home.url.text})</
                            property>
                    </element>
```

```
34                </sec>
35            </layout>
36
37            <element id="web" type="SimpleWeb">
38                <property type="height">420</property>
39                <property type="url">http://www.google.es</property>
40        </element>
41
42      </content>
43    </page>
44  </app>
```

Figura D.4: **xApp Web Browser** -

D.5 Publicar en Twitter

Esta xApp muestra como editar los colores de botones y fondo para recrear el estilo de Twitter, además hace uso de un TextField y un botón con la acción preparada para enviar un Tweet.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>

<app>

  <page id="home" cached="YES">

        <header>
            <property type="title">Quick Tweet</property>
            <property type="color">#4099FF</property>
        </header>

        <content>

            <property type="gradient">#FFFFFF-#4099FF</property>

            <element id="10" type="SimpleSpace">
                <property type="height">20</property>
            </element>

            <element id="tweetField" type="SimpleTextField">
                <property type="margin">20</property>
                <property type="placeholder">Write something</property>
            </element>

            <element id="10" type="SimpleSpace">
                <property type="height">10</property>
            </element>

            <element id="button" type="SimpleButton">
                <property type="margin">20</property>
                <property type="title">Send Tweet</property>
                <property type="color">#4099FF</property>
                <property type="action">
                    home:twitter({var:home.tweetField.text});
                </property>
```

```
36              </element>
37
38          </content>
39
40          <footer>
41          </footer>
42      </page>
43
44  </app>
```

Figura D.5: **xApp Enviar Tweets** -